From Pathwidth to Connected Pathwidth*

Dariusz Dereniowski†

Department of Algorithms and System Modeling,
Gdańsk University of Technology, Poland,
email: deren@eti.pg.gda.pl

Abstract

It is proven that the connected pathwidth of any graph G is at most $2\cdot\mathrm{pw}(G)+1$, where $\mathrm{pw}(G)$ is the pathwidth of G. The method is constructive, i.e. it yields an efficient algorithm that for a given path decomposition of width k computes a connected path decomposition of width at most $2k+1$. The running time of the algorithm is $O(dk^2)$, where d is the number of 'bags' in the input path decomposition.

The motivation for studying connected path decompositions comes from the connection between the pathwidth and the search number of a graph. One of the advantages of the above bound for connected pathwidth is an inequality $\mathtt{cs}(G) \leq 2\mathtt{s}(G)+3$, where $\mathtt{cs}(G)$ and $\mathtt{s}(G)$ are the connected search number and the search number of G. Moreover, the algorithm presented in this work can be used to convert a given search strategy using k searchers into a (monotone) connected one using $2k+3$ searchers and starting at an arbitrary homebase.

Keywords: connected pathwidth, connected searching, fugitive search games, graph searching, pathwidth
AMS subject classifications: 05C83, 68R10, 05C85

*An extended abstract of this work appeared in the proceedings of the 28th Symposium on Theoretical Aspects of Computer Science (STACS) 2011.

†Partially supported by MNiSW grant N N206 379337 (2009-2011).

1 Introduction

The notions of pathwidth and treewidth are receiving increasing interest since the series of Graph Minor articles by Robertson and Seymour, starting with [24]. The importance of those parameters is due to their numerous practical applications, connections with several graph parameters and usefulness in designing graph algorithms. Informally speaking, the *pathwidth* of a graph G, denoted by $\mathrm{pw}(G)$, says how closely G is related to a path. Moreover, a path decomposition captures the linear path-like structure of G. (For a definition see Section 2.)

Here we briefly describe a graph searching game that is one of the main motivations for the results presented in this paper. A team of k searchers is given and the goal is to capture an invisible and fast fugitive located in a given graph G. The fugitive also has the complete knowledge about the graph and about the strategy used by the searchers, and therefore he will avoid being captured as long as possible. The fugitive is captured when a searcher reaches his location. In this setting the game is equivalent to the problem of clearing all edges of a graph that is initially entirely contaminated. There are two main types of this graph searching problem. In the *node searching* two moves are allowed: placing a searcher on a vertex and removing a searcher from a vertex. An edge becomes clear whenever both of its endpoints are simultaneously occupied by searchers. In the *edge searching* we have, besides to the two mentioned moves, a move of sliding a searcher along an edge. In this model an edge $\{u, v\}$ becomes clear if a searcher slides from u to v and either all other edges incident to u have been previously cleared, or another searcher occupies u. In both models the goal is to find a search strategy (a sequence of moves of the searchers) that clears all the edges of G. The *node* (*edge*) *search number* of G, denoted by $\mathtt{ns}(G)$ ($\mathtt{s}(G)$, respectively), equals the minimum number of searchers sufficient to construct a node (edge, respectively) search strategy. An important property is that $\mathrm{pw}(G) = \mathtt{ns}(G) - 1$ for any graph G [15, 16, 17, 21]. The edge searching problem is closely related to node searching, i.e. $|\mathtt{s}(G) - \mathtt{ns}(G)| \leq 1$ [5], and consequently to pathwidth, $\mathrm{pw}(G) \leq \mathtt{s}(G) \leq \mathrm{pw}(G) + 2$.

In this work we are interested in special types of path decompositions called connected path decompositions. The motivation comes from the need of constructing connected search strategies. An edge search strategy is *connected* if the subgraph of G that is clear is always connected. The minimum number of searchers sufficient to construct a connected (edge) search strategy, denoted by $\mathtt{cs}(G)$, is the *connected search number* of G. This model of graph searching receives recently growing interest, because in many applications the connectedness is a requirement.

The concept of recontamination plays an important role in the field of graph searching problems. If the fugitive is able to reach an edge that has been previously cleared, then we say that the edge becomes *recontaminated*. If no recontamination occurs during a search strategy, then the strategy is *monotone*. The minimum number of searchers needed to construct a monotone edge (node, or connected, respectively) search strategy is denoted by $\mathtt{ms}(G)$ ($\mathtt{mns}(G)$, $\mathtt{mcs}(G)$, respectively). For most graph searching models it is proven that there exists a monotone search strategy using the minimum number of searchers, in particular $\mathtt{s}(G) = \mathtt{ms}(G)$ [6, 18], which carries over to node searching, $\mathtt{ns}(G) = \mathtt{mns}(G)$ [17] for any graph G. In the case of connected graph searching problem it turns out that 'recontamination does help' to search a graph [26], that is, there exist graphs G for which each monotone search strategy requires more searchers than some non-monotone search strategies [26], i.e. $\mathtt{mcs}(G) > \mathtt{cs}(G)$. For surveys on graph decompositions and graph searching problems see e.g. [1, 5, 7, 10].

1.1 Related work

There are several results that give a relation between the connected and the 'classical' search numbers of a graph. Fomin et al. proved in [9] that the connected search number of an n-node graph of branchwidth b is bounded by $O(b \log n)$ and this bound is tight. One of the implications of this result is that $\mathtt{cs}(G) = O(\log n)\mathrm{pw}(G)$. Nisse proved in [22] that $\mathtt{cs}(G) \leq (\mathrm{tw}(G)+2)(2\mathtt{s}(G)-1)$ for any chordal graph G. Barrière et al. obtained in [2] a constant upper bound for trees, namely for each tree T, $\mathtt{cs}(T)/\mathtt{s}(T) < 2$. On the other hand, there exists an infinite family of graphs G_k such that $\mathtt{cs}(G_k)/\mathtt{s}(G_k)$ approaches 2 when k goes to infinity [4]. In this work we improve the previously-known bounds for general graphs by proving that

$\mathtt{cs}(G) \leq \mathtt{s}(G) + 3$ for any graph G.

Fraigniaud and Nisse presented in [11] a $O(nk^3)$-time algorithm that takes a width k tree decomposition of a graph and returns a connected tree decomposition of the same width. (For definition of treewidth see e.g. [7, 25].) Therefore, $\mathrm{tw}(G) = \mathrm{ctw}(G)$ for any graph G. That result also yields an upper bound of $\mathtt{cs}(G) \leq (\log n + 1)\mathtt{s}(G)$ for any graph G.

The problems of computing the pathwidth (the search number) and the connected pathwidth (the connected search number) are NP-hard, also for several special classes of graphs, see e.g. [8, 13, 14, 19, 20, 23].

1.2 This work

This paper presents an efficient algorithm that takes a (connected) graph G and its path decomposition $\mathcal{P} = (X_1, \ldots, X_d)$ of width k as an input and returns a connected path decomposition $\mathcal{C} = (Z_1, \ldots, Z_m)$ of width at most $2k + 1$. The running time of the algorithm is $O(dk^2)$ and the number of bags in the resulting path decomposition $\mathcal{C}$ is $m \leq kd$. This solves an open problem stated in several papers, e.g. in [3, 4, 9, 10, 11, 12, 26], since it implies that for any graph G, $\mathrm{cpw}(G) \leq 2\mathrm{pw}(G) + 1$, and improves previously known estimations [9, 22]. The path decomposition $\mathcal{C}$ computed by the algorithm can be used to obtain a monotone connected search strategy using at most $2k + 3$ searchers. Thus, in terms of the graph searching terminology, the above bound immediately implies that $\mathtt{mcs}(G) \leq \mathrm{cpw}(G)+2 \leq 2\mathrm{pw}(G)+3 \leq 2\mathtt{s}(G)+3$. Since $\mathtt{cs}(G) \leq \mathtt{mcs}(G)$, the bound can be restated for the connected search number of a graph, $\mathtt{cs}(G) \leq 2\mathtt{s}(G) + 3$. Moreover, the factor 2 in the bound is tight [4]. The bound can also be used to design approximation algorithms, for it implies that the pathwidth and the connected pathwidth (the search number, the connected search number, and some other search numbers not mentioned here, e.g. the internal search number) are within a constant factor of each other. We can also use the algorithm to construct a (monotone) connected search strategy for $2k + 3$ searchers given, besides of G and $\mathcal{P}$, an input homebase vertex.

2 Preliminaries and basic definitions

Given a simple graph $G = (V(G), E(G))$ and its subset of vertices $X \subseteq V(G)$, the subgraph of G *induced* by X is

$$G[X] = (X, \{\{u, v\} \in E(G) \colon u, v \in X\}).$$

For a simple (not necessary connected) graph G, H is a connected component of G if H is connected, that is, there exists a path in H between each pair of vertices, and each proper supergraph of H is not a subgraph of G. For $X \subseteq V(G)$ let

$$N_G(X) = \{u \in V(G) \setminus X \colon \{u, x\} \in E(G) \text{ for some } x \in X\}.$$

Definition 1 A *path decomposition* of a simple graph $G = (V(G), E(G))$ is a sequence $\mathcal{P} = (X_1, \ldots, X_d)$, where $X_i \subseteq V(G)$ for each $i = 1, \ldots, d$, and

- $\bigcup_{i=1,\ldots,d} X_i = V(G)$,
- for each $\{u, v\} \in E(G)$ there exists $i \in \{1, \ldots, d\}$ such that $u, v \in X_i$,
- for each i, j, k, $1 \leq i \leq j \leq k \leq d$ it holds $X_i \cap X_k \subseteq X_j$.

The *width* of the path decomposition $\mathcal{P}$ is $\mathrm{width}(\mathcal{P}) = \max_{i=1,\ldots,d} |X_i| - 1$. The *pathwidth* of G, $\mathrm{pw}(G)$, is the minimum width over all path decompositions of G.

A path decomposition $\mathcal{P}$ is *connected* if $G[X_1 \cup \cdots \cup X_i]$ is connected for each $i = 1, \ldots, d$. We use the symbol $\mathrm{cpw}(G)$ to denote the minimum width over all connected path decompositions of G.

Definition 2 Given a graph G and its path decomposition $\mathcal{P} = (X_1, \ldots, X_d)$, a node-weighted graph $\mathcal{G} = (V(\mathcal{G}), E(\mathcal{G}), \omega)$ *derived* from G and $\mathcal{P}$ is the graph with vertex set

$$V(\mathcal{G}) = V_1 \cup \cdots \cup V_d,$$

where $V_i = \{v_i(H)\colon H \text{ is a connected component of } G[X_i]\}$, $i = 1, \ldots, d$, and edge set

$$E(\mathcal{G}) = \{\{v_i(H), v_{i+1}(H')\}\colon v_i(H) \in V_i, v_{i+1}(H') \in V_{i+1}, i \in \{1, \ldots, d-1\}, \text{ and } V(H) \cap V(H') \neq \emptyset\}.$$

The weight of a vertex $v_i(H) \in V(\mathcal{G})$, $i \in \{1, \ldots, d\}$, is $\omega(v_i(H)) = |V(H)|$. The *width* of $\mathcal{G}$, denoted by $\text{width}(\mathcal{G})$, equals $\text{width}(\mathcal{P}) + 1$.

In the following we omit a subgraph H of G and the index $i \in \{1, \ldots, d\}$ whenever they are not important when referring to a vertex of $\mathcal{G}$ and we write v instead of $v_i(H)$. For brevity, $\omega(X) = \sum_{x \in X} \omega(x)$ for any subset $X \subseteq V(\mathcal{G})$.

Figures 1(a) and 1(b) present a graph G and its path decomposition $\mathcal{P}$, respectively, where the subgraph structure in each bag X_i is also given. Figure 1(c) depicts the derived graph $\mathcal{G}$. Note that $\mathcal{P}$ is not connected: the subgraphs $G[X_1 \cup \cdots \cup X_i]$ are not connected for $i = 2, 3, 4$.

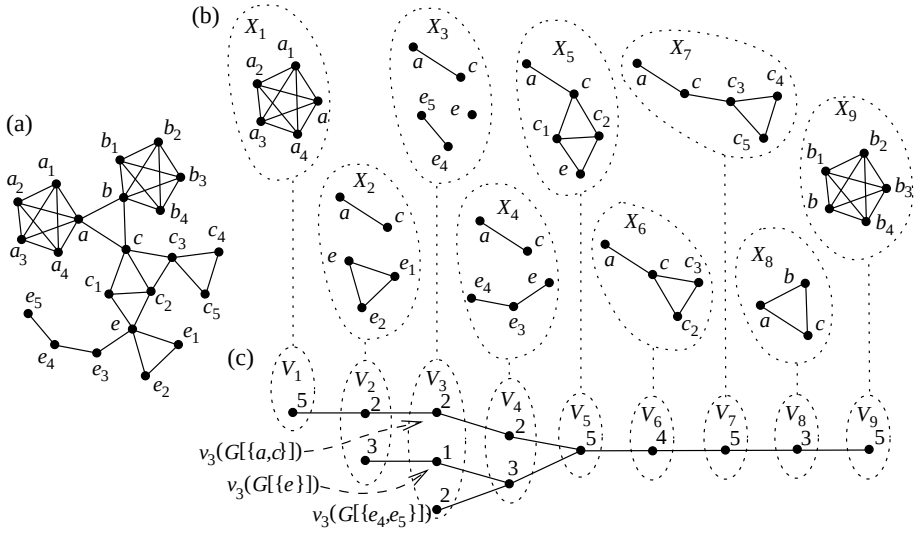

Figure 1: (a) a graph G; (b) a path decomposition $\mathcal{P}$ of G; (c) the weighted graph $\mathcal{G}$ derived from G and $\mathcal{P}$

Let $C \subseteq V(\mathcal{G})$. The *border* $\delta(C)$ of the set C is its subset consisting of all the vertices $v \in C$ such that there exists $u \in V(\mathcal{G}) \backslash C$ adjacent to v in $\mathcal{G}$, i.e. $\delta(C) = N_{\mathcal{G}}(V(\mathcal{G}) \backslash C)$. The algorithms presented in Sections 3 and 5 maintain, besides a set C and its border $\delta(C)$, a partition $\delta_{\mathrm{L}}(C), \delta_{\mathrm{R}}(C)$ such that $\delta(C) = \delta_{\mathrm{L}}(C) \cup \delta_{\mathrm{R}}(C)$ and

$$\delta_{\mathrm{L}}(C) \subseteq V_1 \cup \cdots \cup V_i, \quad \delta_{\mathrm{R}}(C) \subseteq V_{i+1} \cup \cdots \cup V_d, \tag{1}$$

for some $0 \leq i \leq d$. We prove that the partitions used by the algorithms do satisfy the above condition and for the time being we continue with the assumption that for each C and $\delta(C)$ such a partition is given.

Given a set $X \subseteq V(\mathcal{G})$, $X \neq \emptyset$, we define the *left* (*right*) *extremity* of X as $l(X) = \min\{i\colon V_i \cap X \neq \emptyset\}$ ($r(X) = \max\{i\colon V_i \cap X \neq \emptyset\}$, respectively). Note that (1) in particular implies $r(\delta_{\mathrm{L}}(C)) < l(\delta_{\mathrm{R}}(C))$.

3 A simple conversion algorithm

In this section we present a simple algorithm that takes G and a path decomposition $\mathcal{P}$ of G as an input, and returns a connected path decomposition $\mathcal{C}$ of G. However, it is not guaranteed that $\text{width}(\mathcal{C})/\text{width}(\mathcal{P})$

is bounded by a constant. On the other hand, both this one and our final algorithm presented in Section 5 can be seen as the sequences of executions of the basic steps, and the difference lies in using different criteria while constricting those sequences.

The first computation performed by SCP (*Simple Connected Pathwidth*) is the construction of the derived graph $\mathcal{G}$ and in the subsequent steps the algorithm works on $\mathcal{G}$. (Also, most parts of our analysis use $\mathcal{G}$ rather than G.) The algorithm computes a sequence of sets $C_j \subseteq V(\mathcal{G})$, $j = 1, \ldots, m$, called *expansions*. In addition to that, the sets $A_j \subseteq V(\mathcal{G})$, $j = 2, \ldots, m$, and $B_j \subseteq V(\mathcal{G})$, $j = 1, \ldots, m$, are computed. The former one consists of the vertices that are added to C_{j-1} to obtain C_j, while B_j is used to determine the vertices of G that belong to the j-th bag of the resulting path decomposition $\mathcal{C}$. Informally speaking, A_j consists of some vertices in $N_{\mathcal{G}}(C_{j-1})$, and $B_j = \delta(C_j) \cup A_j$, $j = 2, \ldots, m$. The expansion C_1 consists of any vertex in V_1, and $C_m = V(\mathcal{G})$ at the end of the execution of SCP. Moreover, $C_j \subsetneq C_{j+1}$ for each $j = 1, \ldots, m-1$. This guarantees that the final path decomposition obtained from $B_1, \ldots, B_m$ is valid and connected, as proven in Lemma 2. By construction, $\omega(B_j)$ is the size of the corresponding j-th bag of $\mathcal{C}$, but we do not attempt to bound the width of the path decomposition returned by SCP. In this section, besides of the statement of the algorithm, we prove its correctness, i.e. that it stops and returns a connected path decomposition.

The algorithm computes for each expansion C_j two disjoint sets called the *left* and *right borders* of C_j (introduced informally in Section 2), denoted by $\delta_{\mathrm{L}}(C_j)$ and $\delta_{\mathrm{R}}(C_j)$, respectively. It is guaranteed that $\delta_{\mathrm{L}}(C_j) \cup \delta_{\mathrm{R}}(C_j) = \delta(C_j)$ for each $j = 1, \ldots, m$. As it is proven later, the left and right borders are special types of partitions of $\delta(C_j)$. In particular, for each $j = 1, \ldots, m$ there exists an integer $i \in \{0, \ldots, d\}$ such that the left border $\delta_{\mathrm{L}}(C_j)$ is contained in $V_1 \cup \cdots V_i$ and the right border $\delta_{\mathrm{R}}(C_j)$ is a subset of $V_{i+1} \cup \cdots \cup V_d$. For brevity let $l(\delta_{\mathrm{L}}(C_j)) = r(\delta_{\mathrm{L}}(C_j)) = 0$ if $\delta_{\mathrm{L}}(C_j) = \emptyset$ and $l(\delta_{\mathrm{R}}(C_j)) = r(\delta_{\mathrm{R}}(C_j)) = d + 1$ if $\delta_{\mathrm{R}}(C_j) = \emptyset$, where C_j is any expansion. The above-mentioned properties are not needed while proving the correctness of the algorithms, but are necessary for proving the upper bound on width($\mathcal{C}$), where $\mathcal{C}$ is returned by the algorithm from Section 5. For this reason their formal statements and proofs are postponed till Section 6.

As mentioned earlier, both of our algorithms use two basic steps, called LE (*Left Extension*) and RE (*Right Extension*). We describe them first and then we give the pseudo-code of SCP. The steps use m, A_m, B_m, C_m, the borders $\delta_{\mathrm{L}}(C_m)$ and $\delta_{\mathrm{R}}(C_m)$, and the derived graph $\mathcal{G}$ as global variables. Both steps are symmetric, they increment m and compute the above list of sets for the new index m. The input is an integer $i \in \{1, \ldots, d\}$. Informally speaking, the new set C_m is computed by adding to C_{m-1} the selected vertices among those in V_{i-1} in case of Step LE and in V_{i+1} in case of RE that are adjacent to the vertices in C_{m-1}.

Step LE (*Left Extension*)

Input: An integer $i \in \{1, \ldots, d\}$. ($\mathcal{G}$, m, A_m, B_m, C_m, and the borders are used as global variables).

begin

Increment m.

$A_m := V_{i-1} \cap (N_{\mathcal{G}}(\delta(C_{m-1}) \cap V_i) \setminus C_{m-1})$,

$C_m := C_{m-1} \cup A_m$,

$B_m := \delta(C_m) \cup A_m$,

$\delta_{\mathrm{L}}(C_m) := (\delta_{\mathrm{L}}(C_{m-1}) \cup A_m) \cap \delta(C_m)$,

$\delta_{\mathrm{R}}(C_m) := \delta_{\mathrm{R}}(C_{m-1}) \cap \delta(C_m)$,

end Step LE.

Step RE (*Right Extension*)

Input: An integer $i \in \{1, \ldots, d\}$. ($\mathcal{G}$, m, A_m, B_m, C_m, and the borders are used as global variables).

begin

Increment m.

$A_m := V_{i+1} \cap (N_{\mathcal{G}}(\delta(C_{m-1}) \cap V_i) \setminus C_{m-1})$,

$C_m := C_{m-1} \cup A_m$,

$B_m := \delta(C_m) \cup A_m$,

$\delta_{\mathrm{R}}(C_m) := (\delta_{\mathrm{R}}(C_{m-1}) \cup A_m) \cap \delta(C_m)$,

$\delta_{\mathrm{L}}(C_m) := \delta_{\mathrm{L}}(C_{m-1}) \cap \delta(C_m)$,

end Step RE.

Figure 2(a) depicts an exemplary subgraph $\mathcal{G}$ (induced by $V_1 \cup \cdots \cup V_5$) together with white vertices in an expansion C_m. In all cases (including the following figures) $\diamondsuit$ and $\square$ are used to denote the vertices of the right and left borders, respectively. Figures 2(b) and 2(c) show the result of the execution of LE($r(\delta_{\mathrm{L}}(C_m))$) and RE($r(\delta_{\mathrm{L}}(C_m))$), respectively, i.e. the corresponding expansion C_{m+1}. (Note that $r(\delta_{\mathrm{L}}(C_m)) = 3$ in this example.) We always use LE and RE with input $i \in \{r(\delta_{\mathrm{L}}(C_m)), l(\delta_{\mathrm{R}}(C_m))\}$.

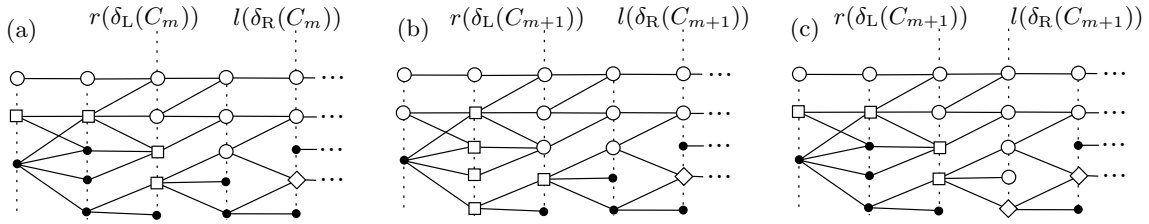

Figure 2: (a) an expansion C_m; (b) C_{m+1} after the execution of $\texttt{LE}(r(\delta_{\mathrm{L}}(C_m)))$; (c) C_{m+1} after the execution of $\texttt{RE}(r(\delta_{\mathrm{L}}(C_m)))$

Procedure SCP (*Simple Connected Pathwidth*)

Input: A simple graph G and a path decomposition $\mathcal{P}$ of G.

Output: A connected path decomposition $\mathcal{C}$ of G.

begin

Use G and $\mathcal{P}$ to calculate the derived graph $\mathcal{G}$. Let v be any vertex in V_1 and let $B_1 = C_1 = \{v\}$ and $\delta_{\mathrm{L}}(C_1) = \emptyset$, $\delta_{\mathrm{R}}(C_1) = \{v\}$. Let $m = 1$. If v has no neighbors in $\mathcal{G}$, then return $\mathcal{P}$.

while $C_m \neq V(\mathcal{G})$ **do**

Execute any of the following Steps S1-S4 that result in computing C_m such that $C_m \neq C_{m-1}$:

S1: Call $\texttt{LE}(r(\delta_{\mathrm{L}}(C_m)))$.

S2: Call $\texttt{RE}(l(\delta_{\mathrm{R}}(C_m)))$.

S3: Call $\texttt{RE}(r(\delta_{\mathrm{L}}(C_m)))$.

S4: Call $\texttt{LE}(l(\delta_{\mathrm{R}}(C_m)))$.

end while

Let $Z_j = \bigcup_{v(H)\in B_j} V(H)$ for each $j = 1, \ldots, m$. **Return** $\mathcal{C} = (Z_1, \ldots, Z_m)$.

end procedure SCP.

First we briefly discuss the initialization stage of CP. The first expansion C_1 consists of a single vertex $v \in V_1$. If v has no neighbors in $\mathcal{G}$, then due to the connectedness of G, $\mathcal{P}$ consists of a single bag, and therefore $\mathcal{P}$ is connected itself.

The instructions prior to the while loop of SCP compute the first set of variables, i.e. for $m = 1$. In the following, one *iteration* of SCP means one iteration of its 'while' loop, which reduces to the execution of one of Steps S1-S4. We discuss Steps S1 and S3, because the other ones are symmetric (S2 is symmetric to S1 and S3 is symmetric to S4). Let us consider Step S1. (We refer here to C_m from the beginning of the execution of Step LE). If no vertex in $V_{r(\delta_{\mathrm{L}}(C_m))} \cap \delta_{\mathrm{L}}(C_m)$ has a neighbor in $V_{r(\delta_{\mathrm{L}}(C_m))+1} \setminus C_m$, then the execution of Step LE guarantees that the right extremity of the left border that will be obtained in LE is strictly less than $r(\delta_{\mathrm{L}}(C_m))$, i.e. $r(\delta_{\mathrm{L}}(C_{m+1})) < r(\delta_{\mathrm{L}}(C_m))$. The right border of the new expansion computed by LE in Step S1 always is equal to the right border of the previous expansion. Note that if no vertex in $V_{r(\delta_{\mathrm{L}}(C_m))} \cap \delta_{\mathrm{L}}(C_m)$ has a neighbor in $V_{r(\delta_{\mathrm{L}}(C_m))-1} \setminus C_m$, then LE called in Step S1 would compute C_{m+1} that is equal to C_m and therefore Step S1 is not executed in such case. Step S3, on the other hand, guarantees that the left border of the new expansion is contained in the left border of the previous one. Then, informally speaking, the right border of the new expansion, that is $\delta_{\mathrm{R}}(C_{m+1})$, besides some vertices from $\delta_{\mathrm{R}}(C_m)$, consists of some vertices in $V_{r(\delta_{\mathrm{L}}(C_m))+1}$.

Now we give one preliminary lemma and then we prove that $\mathcal{C}$ returned by SCP is a connected path decomposition of G.

Lemma 1 *$\mathcal{G}[C_j]$ is connected for each $j = 1, \ldots, m$.*

Proof: By induction on $j \in \{1, \ldots, m\}$. C_1 is connected, because it consists of a single vertex of $\mathcal{G}$. Suppose that C_j, $1 \leq j < m$, is connected and let us consider C_{j+1}. The computation of the latter one is performed in Step `LE` or Step `RE`. By the definition, $C_{j+1} = C_j \cup A_{j+1}$ for some $i \in \{1, \ldots, d\}$. The connectedness of $\mathcal{G}[C_{j+1}]$ follows from the definition of $N_{\mathcal{G}}$ and from the fact that $A_{j+1} \subseteq N_{\mathcal{G}}(C_j)$. □

Lemma 2 *Given a simple graph G and its path decomposition $\mathcal{P} = (X_1, \ldots, X_d)$, `SCP` returns a connected path decomposition $\mathcal{C} = (Z_1, \ldots, Z_m)$ of G.*

Proof: Note that $C_m = V(\mathcal{G})$. This follows from an observation that in each iteration of `SCP` at least one of Steps S1-S4 guarantees to compute C_j such that $C_j \neq C_{j-1}$. Indeed, if $\delta_{\mathrm{L}}(C_{j-1}) \neq \emptyset$, then a vertex $x \in V_i \cap \delta_{\mathrm{L}}(C_{j-1})$, $i = r(\delta_{\mathrm{L}}(C_{j-1}))$, has a neighbor v in either $V_{i-1} \setminus C_{j-1}$ or in $V_{i+1} \setminus C_{j-1}$. By the formulation of Steps `LE` and `RE` we obtain that $v \in C_j$ if Step S1 or Step S3, respectively, are performed. An analogous argument holds for the right border. Note that $\delta_{\mathrm{L}}(C_j) = \delta_{\mathrm{R}}(C_j) = \emptyset$ is, by the connectedness of G, equivalent to $C_j = V(\mathcal{G})$. Thus, the execution of `SCP` stops and the algorithm returns $\mathcal{C} = (Z_1, \ldots, Z_m)$.

Now we prove that $\mathcal{C}$ is a path decomposition of G. Let u be any vertex of G. Since $\mathcal{P}$ is a path decomposition, $u \in X_i$ for some $i \in \{1, \ldots, d\}$. Therefore, u is a vertex of a subgraph H such that $v_i(H) \in V_i$. Since $C_m = V(\mathcal{G})$ and $C_j \subseteq C_{j+1}$ for each $j = 1, \ldots, m-1$, we obtain that there exists a minimum integer $j \in \{1, \ldots, m\}$ such that $v_i(H) \in C_j$. By construction, $v_i(H) \in A_j$ and therefore $v_i(H) \in B_j$. Thus, $u \in Z_j$.

Similarly, for each $\{u, v\} \in E(G)$ there exists $j \in \{1, \ldots, m\}$ such that $u, v \in Z_j$. Indeed, $u, v \in X_i$ for some $i \in \{1, \ldots, d\}$ implies, as before, that $\{u, v\}$ is an edge of some subgraph H of G such that $v_i(H) \in B_j$.

Let i, k be integers, $1 \leq i \leq k \leq m$, and suppose that $u \in Z_i \cap Z_k$. We show that $u \in Z_j$ for each $i \leq j \leq k$. First note that the subgraph $\mathcal{G}[U]$, where $U = \{v_i(H) \colon u \in V(H), i = 1, \ldots, d\}$, is connected, because, if $u \in V(H)$, $v_s(H) \in V_s$ and $u \in V(H')$, $v_{s'}(H) \in V_{s'}$, $s \leq s'$, then by the definition of the derived graph $u \in X_s$ and $u \in X_{s'}$. Since $\mathcal{P}$ is a path decomposition, $u \in X_p$ for each $p = s, \ldots, s'$. Therefore, for each $p = s, \ldots, s'$, there exists a vertex $v_p(H''_p)$ in V_p such that $u \in V(H''_p)$. By the definition, $v_p(H''_p) \in U$. Since $C_j \subseteq C_{j+1}$ and the Steps `LE` and `RE` compute C_{j+1} by taking the vertices in C_j and the subset of $N_{\mathcal{G}}(C_j)$, $j = 1, \ldots, m-1$, the connectedness of $\mathcal{G}[U]$ gives us that $u \in Z_j$ for $i \leq j \leq k$.

Finally we prove that $G[Z_1 \cup \cdots \cup Z_j]$ is connected for each $j = 1, \ldots, m$. Let $u, u' \in Z_1 \cup \cdots \cup Z_j$. From the formulation of Steps `LE` and `RE` it follows that $C_j = B_1 \cup \cdots \cup B_j$. Thus, C_j consists of the vertices $v_i(H)$ such that H is a connected component of $G[Z_p]$, where $p \in \{1, \ldots, j\}$. By the definition of derived graph, there exist two vertices $v_i(H)$, $v_{i'}(H')$ of $\mathcal{G}$ such that $u \in V(H)$, $u' \in V(H')$ and $v_i(H) \in C_j$, $v_{i'}(H') \in C_j$. By Lemma 1, there exists a path P in $\mathcal{G}[C_j]$ connecting $v_i(H)$ and $v_{i'}(H')$. Let the consecutive vertices of P be $v_i(H) = v_{i_1}(H_1), \ldots, v_{i_p}(H_p) = v_{i'}(H')$. By the definition of $\mathcal{G}$, H_s and H_{s+1} share a vertex of G, $s = 1, \ldots, p-1$. Moreover, H_s is connected for each $s = 1, \ldots, p$. This implies that there exists a path P' in G between each vertex of H and each vertex of H', in particular between u and u', and $V(P') \subseteq \bigcup_{1 \leq s \leq p} V(H_s)$. Since $v_i(H) \in C_j$ implies $v_i(H) \in B_s$ for some $s \leq j$, and consequently, $V(H) \subseteq Z_s$, we obtain that $V(P') \subseteq Z_1 \cup \cdots \cup Z_j$, which proves the connectedness of $\mathcal{C}$. □

4 The branches

In this section we introduce the concept of *branches*, our main tool for organizing the sequences of consecutive executions of Steps `LE` and `RE`. A path P in $\mathcal{G}$ is *progressive* if $|V(P) \cap V_i| \leq 1$ for each $i = 1, \ldots, d$. Note that a progressive path that connects a vertex in V_i to a vertex in V_j consists of exactly $|i - j|$ edges, or in other words, a progressive path contains exactly one vertex in V_s for each $s = \min\{i, j\}, \ldots, \max\{i, j\}$.

Definition 3 Given $\mathcal{G}$ and $C \subseteq V(\mathcal{G})$, a *left branch* $\mathcal{B}_{\mathrm{L}}(C, i)$, where $i \leq r(\delta_{\mathrm{L}}(C))$ is the subgraph of $\mathcal{G}$ induced by the vertices in $\delta_{\mathrm{L}}(C)$ and by the vertices $v \in V_k \setminus C$, where $i \leq k < r(\delta_{\mathrm{L}}(C))$, connected by a progressive path to a vertex $x \in V_j \cap \delta_{\mathrm{L}}(C)$ for some $k < j$.

A *right branch* $\mathcal{B}_{\mathrm{R}}(C,i)$, where $i \geq l(\delta_{\mathrm{R}}(C))$ is the subgraph of $\mathcal{G}$ induced by the vertices in $\delta_{\mathrm{R}}(C)$ and by the vertices $v \in V_k \setminus C$, where $l(\delta_{\mathrm{R}}(C)) < k \leq i$, connected by a progressive path to a vertex $x \in V_j \cap \delta_{\mathrm{R}}(C)$ for some $j < k$.

Informally speaking, we construct $\mathcal{B}_{\mathrm{L}}(C,i)$ by taking $\delta_{\mathrm{L}}(C)$ and all vertices in $V_i \cup \cdots \cup V_{r(\delta_{\mathrm{L}}(C))}$ achievable from the vertices $u \in \delta_{\mathrm{L}}(C)$ with progressive paths having u as the right endpoint. We sometimes write $\mathcal{B}$ to refer to a branch whenever its 'direction' or C, i are clear from the context.

Let $\mathcal{B} = \mathcal{B}_{\mathrm{L}}(C,i)$, $i \leq r(\delta_{\mathrm{L}}(C))$, ($\mathcal{B} = \mathcal{B}_{\mathrm{R}}(C,i)$, $i \geq l(\delta_{\mathrm{R}}(C))$) be a branch. A vertex v of $\mathcal{B}$ is *external* if $N_{\mathcal{G}}(v) \nsubseteq C \cup V(\mathcal{B})$. The branch $\mathcal{B}$ is *proper* if it has no external vertices in V_j for each $j > i$ ($j < i$, respectively), while $\mathcal{B}$ is *maximal* if it has no external vertices or if it is proper and $\mathcal{B}_{\mathrm{L}}(C, i-1)$ ($\mathcal{B}_{\mathrm{R}}(C, i+1)$, respectively) is not a proper branch. Let $\mathrm{Ext}(\mathcal{B})$ denote the set of all external vertices of $\mathcal{B}$.

Informally speaking, the external vertices of a branch $\mathcal{B}$ are the ones that have neighbors not in $C \cup V(\mathcal{B})$. A left branch $\mathcal{B}_{\mathrm{L}}(C,i)$ is proper if we can 'grow' it from $r(\delta_{\mathrm{L}}(C))$ to i without leaving any external vertices in V_j, $j > i$. Then, the maximality of the branch implies that we cannot grow the branch beyond i, because either $i = 1$ or the new branch would have an external vertex in V_i (in such case this external vertex must have a neighbor in $V_{i+1} \setminus (C \cup V(\mathcal{B}))$).

Figure 3 illustrates the above definitions. (In all cases the branch is distinguished by the dark area.) Let

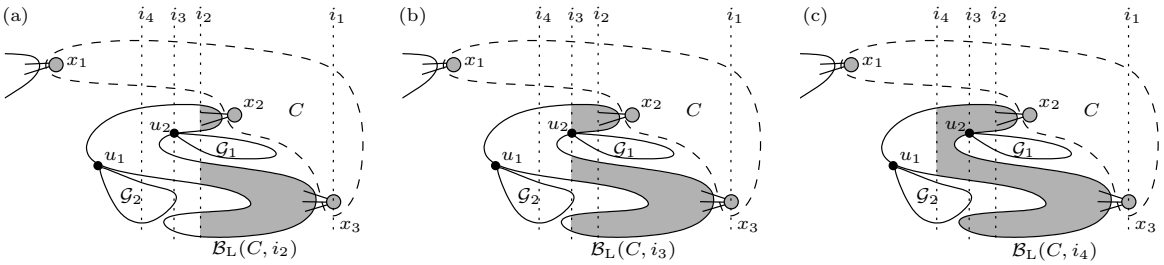

Figure 3: $\mathcal{G}$ with distinguished vertex sets C and $\delta_{\mathrm{L}}(C) = \{x_1, x_2, x_3\}$, and the corresponding (left) branches that are: (a) proper but not maximal; (b) maximal; (c) not proper (thus not maximal)

$\delta_{\mathrm{L}}(C) = \{x_1, x_2, x_3\}$. Figure 3(a) gives $\mathcal{B}_{\mathrm{L}}(C, i_2)$ and this branch is proper, but not maximal for each i_2, $i_3 < i_2 \leq i_1$. The lack of maximality is due to the fact that, informally speaking, we can 'grow' $\mathcal{B}_{\mathrm{L}}(C, i_2)$ by including the corresponding vertices in V_{i_2-1} and the new branch is still proper, as none of its vertices in V_{i_2} are external. The branch $\mathcal{B}_{\mathrm{L}}(C, i_3)$ (see Figure 3(b)) is maximal (thus proper), which follows from the fact that any branch $\mathcal{B}_{\mathrm{L}}(C, i_4)$, where $i_4 < i_3$, is not proper, because it contains an external vertex u_2, as shown in Figure 3(c). Note that the vertices of $\mathcal{G}_1$ and $\mathcal{G}_2$ (except for u_1 and u_2) do not belong to any left branch $\mathcal{B}_{\mathrm{L}}(C,i)$, because they are not connected by progressive paths to x_2 or x_3. In our algorithm we ensure that each branch we use is proper.

An integer j is a *cut* of a left branch $\mathcal{B} = \mathcal{B}_{\mathrm{L}}(C,i)$ if $j \in \{i, \ldots, r(V(\mathcal{B}))\}$. Then, its *weight* equals $\omega(\mathrm{Ext}(\mathcal{B}_{\mathrm{L}}(C,j)))$. An integer j is a *cut* of a right branch $\mathcal{B} = \mathcal{B}_{\mathrm{R}}(C,i)$ if $j \in \{l(V(\mathcal{B})), \ldots, i\}$. Then, its *weight* equals $\omega(\mathrm{Ext}(\mathcal{B}_{\mathrm{R}}(C,j)))$. A cut of minimum weight is a *bottleneck* of a branch.

We finish this section with the following observations.

Observation 1 *For each expansion C it holds $V_j \cap V(\mathcal{B}_{\mathrm{L}}(C,i)) = V_j \cap V(\mathcal{B}_{\mathrm{L}}(C,i'))$ for each $i \leq i' \leq j \leq r(\delta_{\mathrm{L}}(C))$ and $V_j \cap V(\mathcal{B}_{\mathrm{R}}(C,i)) = V_j \cap V(\mathcal{B}_{\mathrm{R}}(C,i'))$ for each $l(\delta_{\mathrm{R}}(C)) \leq j \leq i' \leq i$.* □

Observation 2 *For each expansion C the weight of a cut j of a proper branch $\mathcal{B}_{\mathrm{L}}(C,i)$, $i \leq j \leq r(\delta_{\mathrm{L}}(C))$, is less than or equal to $\omega((V_1 \cup \cdots \cup V_{j-1}) \cap \delta_{\mathrm{L}}(C)) + \omega(V_j \cap V(\mathcal{B}_{\mathrm{L}}(C,i)))$, and the weight of a cut j of a proper branch $\mathcal{B}_{\mathrm{R}}(C,i)$, $l(\delta_{\mathrm{R}}(C)) \leq j \leq i$, is less than or equal to $\omega((V_{j+1} \cup \cdots \cup V_d) \cap \delta_{\mathrm{R}}(C)) + \omega(V_j \cap V(\mathcal{B}_{\mathrm{R}}(C,i)))$.* □

5 The algorithm

We start with two subroutines PLB (*Process Left Branch*) and PRB (*Process Right Branch*) that are used by the main procedure given in this section. The input to PLB and to PRB consists of an integer t, $t \in \{1, \ldots, d\}$. In the following we say for brevity that an expansion has been computed by PLB or PRB whenever it has been computed by LE or RE called by PLB or PRB, respectively. Due to symmetry we skip the informal description of PRB here. If C_{j+1} and $C_{j'}$ are the first and the last expansions computed by PLB, then $C_{j'} = C_j \cup V(\mathcal{B}_{\mathrm{L}}(C_j, t))$, which we formally show in Lemma 6. This is achieved by several executions of Step LE. In particular, LE($r(\delta_{\mathrm{L}}(C_m))$) is repeatedly called as long as the right extremity of the left border of the current expansion is greater than t. The procedures PLB(t) and PRB(t) are as follows.

Procedure PLB (*Process Left Branch*)

Input: An integer t. (m, C_m and $\delta_{\mathrm{L}}(C_m)$ are used as global variables)
begin
  while $r(\delta_{\mathrm{L}}(C_m)) > t$ **do**
    Call LE($r(\delta_{\mathrm{L}}(C_m))$).
end procedure PLB.

Procedure PRB (*Process Right Branch*)

Input: An integer t. (m, C_m and $\delta_{\mathrm{R}}(C_m)$ are used as global variables)
begin
  while $l(\delta_{\mathrm{R}}(C_m)) < t$ **do**
    Call LE($l(\delta_{\mathrm{R}}(C_m))$).
end procedure PRB.

Now we are ready to give the pseudo-code of the main algorithm CP (*Connected Pathwidth*). Its input consists of, as in the case of SCP, a simple graph G and its path decomposition $\mathcal{P}$.

Algorithm CP (*Connected Pathwidth*)

Input: A simple graph G and a path decomposition $\mathcal{P}$ of G.
Output: A connected path decomposition $\mathcal{C}$ of G.
begin
  (*Initialization.*)
    I.1: Use G and $\mathcal{P}$ to calculate the derived graph $\mathcal{G}$. Let v be any vertex in V_1 and let $B_1 = C_1 = \{v\}$ and $\delta_{\mathrm{L}}(C_1) = \emptyset$, $\delta_{\mathrm{R}}(C_1) = \{v\}$. Let $m = 1$. If v has no neighbors in $\mathcal{G}$, then return $\mathcal{P}$.
    I.2: Find the maximal right branch $\mathcal{B}_{\mathrm{R}}(C_1, a_0)$ with a bottleneck a'_0 ($1 \le a'_0 \le a_0$). Call PRB(a'_0).
  (*Main loop.*)
    while $C_m \neq V(\mathcal{G})$ **do**
      if $\omega(\delta_{\mathrm{L}}(C_m)) > \omega(\delta_{\mathrm{R}}(C_m))$ **then**
        L.1: Find the maximal left branch $\mathcal{B}_1 = \mathcal{B}_{\mathrm{L}}(C_m, t_1)$. Call PLB($t_1$).
        L.2: Call RE(t_1).
      else
        R.1: Find the maximal right branch $\mathcal{B}_1 = \mathcal{B}_{\mathrm{R}}(C_m, t_1)$. Call PRB($t_1$).
        R.2: Call LE(t_1).
      end if
      RL.3: Find the maximal right and left branches $\mathcal{B}_2 = \mathcal{B}_{\mathrm{R}}(C_m, t_2)$ and $\mathcal{B}_3 = \mathcal{B}_{\mathrm{L}}(C_m, t_3)$, with bottlenecks t'_2 and t'_3, respectively. Call PLB(t'_3) and PRB(t'_2).
    end while.
    Let $Z_j = \bigcup_{v(H) \in B_j} V(H)$ for each $j = 1, \ldots, m$. **Return** $\mathcal{C} = (Z_1, \ldots, Z_m)$.
end procedure CP.

Note that Step I.1 of CP consists of the instructions from the initialization stage of SCP. Then, the algorithm finds in Step I.2 the maximal right branch 'emanating' from v, and includes into C_1 the part of the branch that 'reaches' a bottleneck of the branch. This is achieved by the call to PRB with the input a'_0.

In the following, one *iteration* of CP, PLB or PRB means one iteration of the 'while' loop in the corresponding procedure. Thus, in the case of CP, one iteration reduces to executing Steps L.1, L.2, RL.3 or R.1, R.2, RL.3, while in the procedures PLB and PRB one iteration results in executing Step LE or Step RE, respectively. We use the symbols $\mathcal{B}_i, t_i$, $i = 1, 2, 3$, and t'_i, $i = 2, 3$, to refer to the variables used in CP. In what follows we denote for brevity $\mathcal{B}'_2 = \mathcal{B}_{\mathrm{R}}(C_m, t'_2)$ and $\mathcal{B}'_3 = \mathcal{B}_{\mathrm{L}}(C_m, t'_3)$. Informally speaking, $\mathcal{B}'_2$ and $\mathcal{B}'_3$ are the branches $\mathcal{B}_2$ and $\mathcal{B}_3$, respectively, restricted to the vertices up to the corresponding cut t'_2 or t'_3. Note that the outcome in Step RL.3 (in terms of the expansion obtained at the end of the iteration) is the same regardless of the order of making the calls to PLB and PRB. Due to the analysis in Section 6, the approximation guarantee of CP remains the same for each order of making those calls in Step RL.3.

The branches are used in the subsequent iterations of CP in the way presented in Figure 4, where the Steps L.1, L2 and RL.3 are shown (the execution of Steps R.1 and R.2 is symmetric with respect to Steps L.1 and L.2). Figure 4(a) gives C together with $\delta_{\mathrm{L}}(C)$ and $\delta_{\mathrm{R}}(C)$. The dark area is the maximal left branch $\mathcal{B}_1$

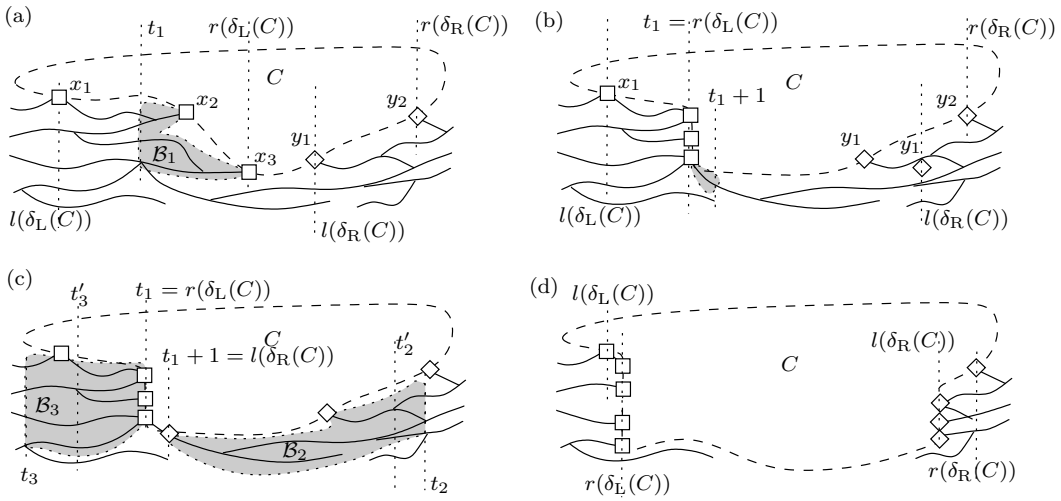

Figure 4: The execution of one iteration of CP (Steps L.1, L.2, RL.3): (a) an expansion C with $\delta_{\mathrm{L}}(C) = \{x_1, x_2, x_2\}$ and $\delta_{\mathrm{R}}(C) = \{y_1, y_2\}$ and the branch $\mathcal{B}_1$ from Step L.1; (b) C at the end of Step L.1, dark area marks the vertices to be included in Step L.2; (c) C at the end of Step L.2, together with $\mathcal{B}_2$ and $\mathcal{B}_3$ computed in Step RL.3; (d) C at the end of the iteration

from Step L.1. Note that if $t_1 = r(\delta_{\mathrm{L}}(C))$, where C is the expansion from the beginning of an iteration, then no new expansion is computed during Step L.1, and the algorithm proceeds to Step L.2. Such a situation occurs if $\mathcal{B}_1$ contains only the vertices in $\delta_{\mathrm{L}}(C)$ or, equivalently, if a vertex in $V_{r(\delta_{\mathrm{L}}(C))} \cap \delta_{\mathrm{L}}(C)$ has a neighbor in $V_{r(\delta_{\mathrm{L}}(C))+1} \setminus C$. The result of the execution of Step L.1 is shown in Figure 4(b) together with dark area marking the nodes in V_{t_1+1} that will be included into the expansion in Step L.2 — see Figure 4(c) for the result of the execution of Step L.2 and for the maximal branches $\mathcal{B}_2$ and $\mathcal{B}_3$ (also, the exemplary integers t'_2 and t'_3 are shown to indicate the corresponding 'sub-branches' $\mathcal{B}'_2$ and $\mathcal{B}'_3$ that reach the minimum cuts of $\mathcal{B}_2$ and $\mathcal{B}_3$, respectively). If none of the vertices included into C in Step L.2 are external, then no new right border vertices in V_{t_1+1} are introduced, and therefore the new expansion in Figure 4(c) has the right border that is a subset of the right border of the expansion in Figure 4(b). Finally, Figure 4(d) gives the result of the execution of RL.3 (and thus the entire iteration).

In this section we prove the correctness of CP, while Section 6 analyzes the width of $\mathcal{C}$ returned by CP. Lemmas 5 and 6 given below demonstrate how the expansions change between the subsequent calls to PLB

and PRB. They show that, informally speaking, if (during the execution of CP) we take a proper branch $\mathcal{B}_{\mathrm{L}}(C_m, i)$ or $\mathcal{B}_{\mathrm{R}}(C_m, i)$ and call PLB(i) or PRB(i), respectively, then the expansion obtained at the end of the execution of PLB(i) or PRB(i), respectively, consists of the vertices of C_m and the vertices of the corresponding branch.

First we introduce the concept of moving the borders and we state two preliminary lemmas. We say that C_j *moves* the right border of C_{j-1} if $l(\delta_{\mathrm{R}}(C_j)) > l(\delta_{\mathrm{R}}(C_{j-1}))$. Similarly, C_j *moves* the left border of C_{j-1} if $r(\delta_{\mathrm{L}}(C_j)) < r(\delta_{\mathrm{L}}(C_{j-1}))$.

Lemma 3 *If $\mathcal{B}_{\mathrm{L}}(C_{j'}, t)$, $t \leq r(\delta_{\mathrm{L}}(C_{j'}))$, (respectively $\mathcal{B}_{\mathrm{R}}(C_{j'}, t)$, $t \geq l(\delta_{\mathrm{R}}(C_{j'}))$) is proper, then each expansion C_j computed by* PLB(t) *(*PRB(t)*) moves the left (right) border of C_{j-1}, where $C_{j'}$ is the expansion from the beginning of the execution of the corresponding procedure* PLB *or* PRB.

Proof: Let C_j be an expansion constructed in an iteration of PRB and the other case is symmetric. In each iteration of PRB the input integer i passed to RE satisfies $i = l(\delta_{\mathrm{R}}(C_{j-1}))$. By the formulation of PRB, $i < t$. Thus, since $\mathcal{B}_{\mathrm{R}}(C_{j'}, t)$ is proper, the vertices in $V_i \cap \delta_{\mathrm{R}}(C_{j-1})$ have no neighbors in $V_{i-1} \setminus C_{j-1}$. Hence, the instructions in Step RE imply that $V_i \cap \delta_{\mathrm{R}}(C_j) = \emptyset$. Therefore, $l(\delta_{\mathrm{R}}(C_j)) > i$, i.e. C_j moves the right border of C_{j-1}. □

The above, and the fact that the branches $\mathcal{B}_i$, $i = 1, 2, 3$, computed by CP are proper, give the following.

Lemma 4 *If C_j, $j \in \{2, \ldots, m\}$, is an expansion calculated in Step* LE *(Step* RE*) invoked by* PLB *(*PRB*, respectively), then C_j moves the left (right, resp.) border of C_{j-1}.* □

Lemma 5 *Let $C_{j'+1}$ and C_j be, respectively, the first and the last expansions computed by the procedure* PLB(t) *or* PRB(t). *If $\mathcal{B}_{\mathrm{L}}(C_{j'}, t)$ is proper and $t \leq r(\delta_{\mathrm{L}}(C_{j'}))$, then the execution of* PLB(t) *results in $C_j = C_{j'} \cup V(\mathcal{B}_{\mathrm{L}}(C_{j'}, t))$. If $\mathcal{B}_{\mathrm{R}}(C_{j'}, t)$ is proper and $t \geq l(\delta_{\mathrm{R}}(C_{j'}))$, then the execution of* PRB(t) *results in $C_j = C_{j'} \cup V(\mathcal{B}_{\mathrm{R}}(C_{j'}, t))$.*

Proof: We consider the call to PLB(t), the proof being analogous in the other case. Suppose that $u \in V_i$ and $v \in V_{i'} \cap \delta_{\mathrm{L}}(C_{j'})$, where $t \leq i \leq i'$, are connected by a progressive path P in $\mathcal{G}$. By the formulation of PLB and by Lemma 3, $V(P) \subseteq C_j$. Thus, by the definition of a branch, $C_{j'} \cup V(\mathcal{B}_{\mathrm{L}}(C_{j'}, t)) \subseteq C_j$. It follows directly from the formulation of PLB that $C_j \subseteq C_{j'} \cup V(\mathcal{B}_{\mathrm{L}}(C_{j'}, t))$. □

Lemma 6 *Let C_{s_0} be an expansion from the beginning of an iteration of* CP*, and let C_{s_i}, $i = 1, 2, 3$, be the expansions obtained at the end of Steps L.1, L.2 and RL.3 or R.1, R.2 and RL.3 in this iteration, respectively. Then, $C_{s_1} = C_{s_0} \cup V(\mathcal{B}_1)$, $C_{s_2} = C_{s_1} \cup A_{s_2}$ and $C_{s_3} = C_{s_2} \cup V(\mathcal{B}'_2) \cup V(\mathcal{B}'_3)$. Moreover, $C_{s_3} \neq C_{s_0}$.*

Proof: The equalities follow directly from Lemma 5 and from the formulation of the algorithm CP.

Since the analysis in the cases $\omega(\delta_{\mathrm{L}}(C_{s_0})) > \omega(\delta_{\mathrm{R}}(C_{s_0}))$ and $\omega(\delta_{\mathrm{L}}(C_{s_0})) \leq \omega(\delta_{\mathrm{R}}(C_{s_0}))$ is similar, we assume that the former occurs. Thus, the Steps L.1, L.2, RL.3 of CP are executed. If $t_1 < r(\delta_{\mathrm{L}}(C_{s_0}))$ in Step L.1, then by construction $V(\mathcal{B}_1) \setminus C_{s_0} \neq \emptyset$ and Lemma 5 implies that $C_{s_3} \neq C_{s_0}$. Otherwise, that is, if $t_1 = r(\delta_{\mathrm{L}}(C_{s_0}))$ in Step L.1, then there exists an external vertex $v \in V_{t_1} \cap C_{s_0}$ adjacent to $u \in V_{t_1+1} \setminus C_{s_0}$. By the formulation of Step L.2 of CP, $u \in A_{s_2}$, and consequently $u \in C_{s_2}$, which gives $C_{s_3} \neq C_{s_0}$. □

Now observe that the procedure CP is a special case of SCP in the sense that if we execute CP for the given G and $\mathcal{P}$ and we take the history of the executions of Steps LE and RE, then, due to the fact that the choices between the steps S.1-S.4 in SCP are arbitrary, it is possible to obtain exactly the same chain of executions of Steps LE and RE in SCP. Moreover, Lemma 6 implies that CP stops and returns $\mathcal{C}$. Therefore, by Lemma 2, we obtain the following.

Lemma 7 *The execution of the procedure* CP *stops, and, for the given input G and $\mathcal{P}$,* CP *returns a connected path decomposition of G.* □

Figure 5 gives an example of the execution of CP. In particular, Figure 5(a) presents a graph $\mathcal{G}$ and C_3 (this is the expansion obtained at the end of initialization of CP, where the vertex v from Step I.1 is the one with the weight 2 in V_1). Figures 5(b)-(d) depict the state of the algorithm at the end of the first three iterations. (The fourth iteration executes Steps L.1, L.2 and RL.3, which ends the computation.)

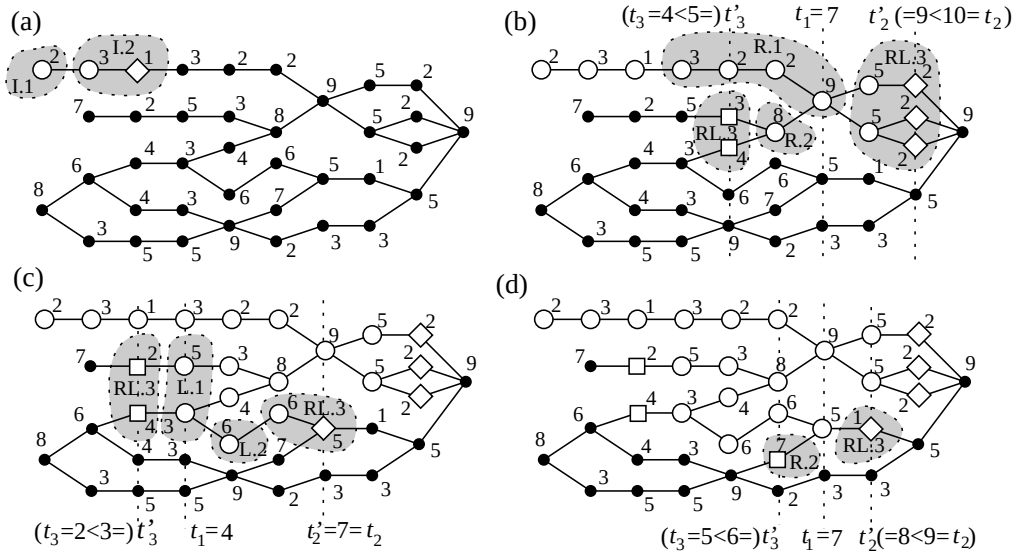

Figure 5: a graph $\mathcal{G}$ (the integers are vertex weights) with white vertices in C_m representing the state of CP after: (a) the initialization; (b) first iteration (Steps R.1, R.2, RL.3); (c) second iteration (Steps L.1, L.2, RL.3); (c) third iteration (Steps R.1, R.2, RL.3); (in all cases the shaded area covers the vertices added to the current expansion during the particular step)

6 The approximation guarantee of the algorithm

In this section we analyze the width of the path decomposition $\mathcal{C}$ calculated by CP for the given G and $\mathcal{P}$. First we introduce the concept of nested expansion, which, informally speaking, is as follows. The first condition for C to be nested states that the weight of $V_i \cap C$ for any i 'between' the right extremity of the left border and the left extremity of the right border (by Lemma 9 the former is less than the latter in each expansion computed by CP) is greater than or equal to the weight of the left or the right border of C. The remaining conditions refer to the situation 'inside' the borders and are symmetric for the left and right borders. Condition (ii) for the left border requires that the weight of $V_i \cap C$, where $i \leq r(\delta_{\mathrm{L}}(C))$, is not less than the weight of the left border restricted to the vertices in $V_1 \cup \cdots \cup V_i$. Finally, condition (iii) gives a 'local' minimality. Suppose that we take any left branch $\mathcal{B}_{\mathrm{L}}(C,i)$ (where i by the definition is $\leq r(\delta_{\mathrm{L}}(C))$) and we add the vertices of this branch to C in the way it is done in procedure PLB, then we 'arrive' at some cut of this branch. Then, (iii) for C guarantees that the weight of the left border of the new expansion, i.e. $\omega(\delta_{\mathrm{L}}(C \cup V(\mathcal{B}_{\mathrm{L}}(C,i))))$, is greater than or equal to the weight of the left border of C.

Formally, we say that an expansion C is *nested* if it satisfies the following conditions:

(i) $\omega(V_i \cap C) \geq \min\{\omega(\delta_{\mathrm{L}}(C)), \omega(\delta_{\mathrm{R}}(C))\}$ for each $i = r(\delta_{\mathrm{L}}(C)), \ldots, l(\delta_{\mathrm{R}}(C))$,

(ii) $\omega(V_i \cap C) \geq \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C))$ for each $i \leq r(\delta_{\mathrm{L}}(C))$, and $\omega(V_i \cap C) \geq \sum_{p \geq i} \omega(V_p \cap \delta_{\mathrm{R}}(C))$ for each $i \geq l(\delta_{\mathrm{R}}(C))$,

(iii) $r(\delta_{\mathrm{L}}(C))$ is a bottleneck of each branch $\mathcal{B}_{\mathrm{L}}(C,i)$, where $i \leq r(\delta_{\mathrm{L}}(C))$, and $l(\delta_{\mathrm{R}}(C))$ is a bottleneck of each branch $\mathcal{B}_{\mathrm{R}}(C,i)$, where $i \geq l(\delta_{\mathrm{R}}(C))$.

Figure 6 presents a subgraph of $\mathcal{G}$ induced by the vertices that belong to an expansion C, and with distinguished left and right borders, $\delta_{\mathrm{L}}(C) = \{x_1, \ldots, x_4\}$, $\delta_{\mathrm{R}}(C) = \{y_1, \ldots, y_4\}$. In this example $r(\delta_{\mathrm{L}}(C)) = i + 5$ and $l(\delta_{\mathrm{R}}(C)) = i + 8$. If this expansion is nested, then it holds in particular: condition (ii) implies $\omega(V_{i'} \cap C) \geq \omega(\{x_1, x_2, x_3\})$ for each $i' = i + 1, \ldots, i + 4$, $\omega(V_{i+5} \cap C) \geq \omega(\{x_1, x_2, x_3, x_4\})$; condition (i) implies $\omega(V_{i+6} \cap C) \geq \min\{\omega(\delta_{\mathrm{L}}(C)), \omega(\delta_{\mathrm{R}}(C))\} = \min\{\omega(\{x_1, \ldots, x_4\}), \omega(\{y_1, \ldots, y_4\})\}$.

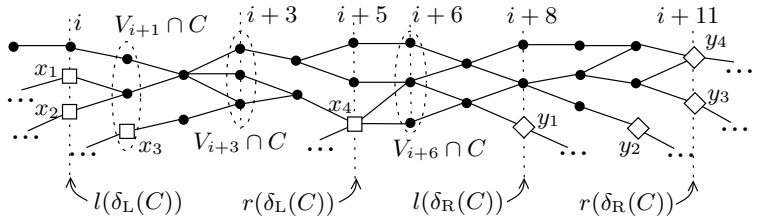

Figure 6: An example of a nested expansion

Not all expansions computed by `CP` are nested, but we prove that all of them satisfy (ii) (see Lemmas 10-13). In particular we argue that if an expansion from the beginning of an iteration of `CP` is nested, then each expansion computed in this iteration satisfies (ii) (Lemma 12). Moreover, an expansion obtained at the end of this iteration is nested (Lemma 13), which allows us to apply an induction on the number of iterations to prove the claim. We also prove that each expansion obtained in Step L.2 and in Step R.2 of `CP` satisfies (i). Those facts are used in Lemma 14 to prove that $\omega(B_j) \leq 2 \cdot \mathrm{width}(\mathcal{G})$ for each $j = 1, \ldots, m$. Note that $\omega(B_j) = |Z_j|$ for each $j = 1, \ldots, m$. Finally, we give the main results in Theorems 1 and 2. We start with two preliminary lemmas that analyze how the borders change while `PLB` and `PRB` execute.

Lemma 8 *$\delta(C_j) = \delta_{\mathrm{L}}(C_j) \cup \delta_{\mathrm{R}}(C_j)$ for each $j = 1, \ldots, m$.*

Proof: $\delta_{\mathrm{L}}(C_j) \cup \delta_{\mathrm{R}}(C_j) \subseteq \delta(C_j)$ follows directly from Steps `LE` and `RE`. To prove that $\delta(C_j) \subseteq \delta_{\mathrm{L}}(C_j) \cup \delta_{\mathrm{R}}(C_j)$ we use induction on j. For $j = 1$ the lemma follows directly from Step I.1 of `CP`.

Suppose that C_j, $1 \leq j < m$, satisfies the hypothesis. Expansion C_{j+1} is constructed in Step `LE` or `RE`. Both cases are analogous so assume that the computation occurs in Step `LE`. Let $x \in \delta(C_{j+1})$. By construction and by the induction hypothesis, $x \in \delta(C_j) \cup A_{j+1} = \delta_{\mathrm{L}}(C_j) \cup \delta_{\mathrm{R}}(C_j) \cup A_{j+1}$. If $x \in \delta_{\mathrm{L}}(C_j) \cup A_{j+1}$, then $x \in \delta_{\mathrm{L}}(C_{j+1})$, because (by the definition) $\delta_{\mathrm{L}}(C_{j+1}) = (\delta_{\mathrm{L}}(C_j) \cup A_{j+1}) \cap \delta(C_{j+1})$ and, by assumption, $x \in \delta(C_{j+1})$. If $x \in \delta_{\mathrm{R}}(C_j)$, then $x \in \delta_{\mathrm{R}}(C_{j+1})$, because $\delta_{\mathrm{R}}(C_{j+1}) = \delta_{\mathrm{R}}(C_j) \cap \delta(C_{j+1})$. □

Lemma 9 *$r(\delta_{\mathrm{L}}(C_j)) < l(\delta_{\mathrm{R}}(C_j))$ for each $j = 1, \ldots, m$.*

Proof: To prove the lemma we use induction on j. The claim clearly holds for $j = 1$, and consequently, by Lemma 4, it holds for all expansions obtained in the initialization stage of `CP`.

Suppose that C_j, $1 \leq j < m$, satisfies the hypothesis. The first case to consider is when the expansion C_{j+1} is constructed in Step `LE` or `RE` called by `PLB` or `PRB`, respectively. Due to symmetry assume that the former occurs. By Lemma 4, C_{j+1} moves the left border of C_j. Thus, by the induction hypothesis, $r(\delta_{\mathrm{L}}(C_{j+1})) < r(\delta_{\mathrm{L}}(C_j)) < l(\delta_{\mathrm{R}}(C_j)) \leq l(\delta_{\mathrm{R}}(C_{j+1}))$. The last inequality is due to $\delta_{\mathrm{R}}(C_{j+1}) \subseteq \delta_{\mathrm{R}}(C_j)$, which follows from the definition, $\delta_{\mathrm{R}}(C_{j+1}) = \delta_{\mathrm{R}}(C_j) \cap \delta(C_{j+1})$.

The second case occurs when C_{j+1} is computed in Step L.2 or R.2 of `CP`. (We consider Step R.2, as the other case is similar.) By construction, $\delta_{\mathrm{L}}(C_{j+1}) \subseteq \delta_{\mathrm{L}}(C_j) \cup V_{t_1 - 1}$. It holds $l(\delta_{\mathrm{R}}(C_j)) \geq t_1$. By the induction hypothesis, $r(\delta_{\mathrm{L}}(C_j)) < l(\delta_{\mathrm{R}}(C_j))$. Thus, $r(\delta_{\mathrm{L}}(C_{j+1})) \leq \max\{r(\delta_{\mathrm{L}}(C_j)), t_1 - 1\} < l(\delta_{\mathrm{R}}(C_j)) \leq l(\delta_{\mathrm{R}}(C_{j+1}))$. □

In order to simplify the statements denote $\widetilde{C} = C \setminus \delta(C)$ for an expansion C. Note that (ii) is equivalent to

(ii') $\omega(V_i \cap \widetilde{C}) \geq \sum_{p<i} \omega(V_p \cap \delta_{\mathrm{L}}(C))$ for each $i \leq r(\delta_{\mathrm{L}}(C))$, and $\omega(V_i \cap \widetilde{C}) \geq \sum_{p>i} \omega(V_p \cap \delta_{\mathrm{R}}(C))$ for each $i \geq l(\delta_{\mathrm{R}}(C))$,

because, by Lemmas 8 and 9, $V_i \cap C$ is the sum of disjoint sets $V_i \cap \widetilde{C}$ and $V_i \cap \delta_{\mathrm{L}}(C)$ when $i \leq r(\delta_{\mathrm{L}}(C))$ and $V_i \cap \widetilde{C}$ and $V_i \cap \delta_{\mathrm{R}}(C)$ when $i \geq l(\delta_{\mathrm{R}}(C))$ for each expansion C.

Lemma 10 *Let $j \in \{2, \ldots, m\}$. If C_{j-1} satisfies* (ii) *and C_j has been computed by the procedure* `PLB` *or* `PRB`*, then C_j satisfies* (ii)*.*

Proof: All expansions C_j obtained in the initialization stage of `CP` satisfy (ii), because $\delta_{\mathrm{L}}(C_j) = \emptyset$ and $\delta_{\mathrm{R}}(C_j)$ is contained in a single set V_i for some $i \in \{1, \ldots, d\}$. Assume in the following without loss of generality that C_j has been computed by `PLB`, that is, in Step `LE` and the proof of the other case is analogous. By construction, $\delta_{\mathrm{R}}(C_j) = \delta_{\mathrm{R}}(C_{j-1}) \cap \delta(C_j) \subseteq \delta_{\mathrm{R}}(C_{j-1})$. Thus, $l(\delta_{\mathrm{R}}(C_j)) \geq l(\delta_{\mathrm{R}}(C_{j-1}))$, and therefore by (ii) for C_{j-1} we obtain that

$$\omega(V_i \cap C_j) = \omega(V_i \cap C_{j-1}) \geq \sum_{p \geq i} \omega(V_p \cap \delta_{\mathrm{R}}(C_{j-1})) \geq \sum_{p \geq i} \omega(V_p \cap \delta_{\mathrm{R}}(C_j)), \tag{2}$$

for each $i \geq l(\delta_{\mathrm{R}}(C_j))$.

Now we prove the condition (ii) for the left border of C_j. By Lemma 4, C_j moves the left border of C_{j-1}. Thus,

$$r(\delta_{\mathrm{L}}(C_j)) \leq r(\delta_{\mathrm{L}}(C_{j-1})) - 1. \tag{3}$$

It follows from `PLB` and from the definition of $\mathcal{G}$ that for each $p < r(\delta_{\mathrm{L}}(C_j)) - 1$ it holds

$$V_p \cap C_j = V_p \cap C_{j-1} \text{ and } V_p \cap \delta_{\mathrm{L}}(C_j) = V_p \cap \delta_{\mathrm{L}}(C_{j-1}), \tag{4}$$

while for $p = r(\delta_{\mathrm{L}}(C_j)) - 1$ we have

$$V_p \cap C_j = V_p \cap C_{j-1} \text{ and } V_p \cap \delta_{\mathrm{L}}(C_j) \subseteq V_p \cap \delta_{\mathrm{L}}(C_{j-1}) \tag{5}$$

(see Figure 7 for a case when $V_p \cap \delta_{\mathrm{L}}(C_j) \neq V_p \cap \delta_{\mathrm{L}}(C_{j-1})$ for $p = r(\delta_{\mathrm{L}}(C_j)) - 1$). Thus, by (ii) for C_{j-1}, and by (3), (4), (5), we obtain that for each $i < r(\delta_{\mathrm{L}}(C_j))$,

$$\omega(V_i \cap C_j) = \omega(V_i \cap C_{j-1}) \geq \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{j-1})) \geq \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C_j)). \tag{6}$$

Let $i = r(\delta_{\mathrm{L}}(C_j))$. It holds $V_i \cap \widetilde{C}_{j-1} \subseteq V_i \cap \widetilde{C}_j$ (see also Figure 7), which gives $\omega(V_i \cap \widetilde{C}_j) \geq \omega(V_i \cap \widetilde{C}_{j-1})$.

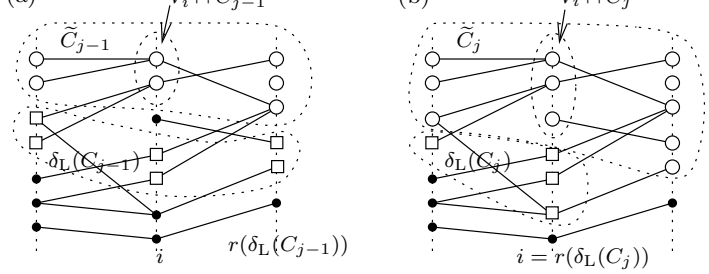

Figure 7: Proof of Lemma 10: the subgraph $\mathcal{G}[V_{i-1} \cup V_i \cup V_{i+1}]$, where $i = r(\delta_{\mathrm{L}}(C_j))$ with white vertices in: (a) C_{j-1}, and (b) C_j

Thus, by (ii'),(4),(5), by (ii) for C_{j-1}, and by Lemma 9,

$$\omega(V_i \cap C_j) = \omega(V_i \cap \widetilde{C}_j) + \omega(V_i \cap \delta_{\mathrm{L}}(C_j)) \geq \omega(V_i \cap \widetilde{C}_{j-1}) + \omega(V_i \cap \delta_{\mathrm{L}}(C_j))$$
$$\geq \sum_{p<i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{j-1})) + \omega(V_i \cap \delta_{\mathrm{L}}(C_j)) \geq \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C_j)). \quad (7)$$

Equations (2), (6) and (7) prove (ii) for C_j. □

Lemma 11 *Let C_{s_0} be the expansion from the beginning of an iteration of* CP *and let C_{s_1} be the expansion obtained at the end of Step L.1 or Step R.1 of this iteration. If C_{s_0} is nested, then C_{s_1} satisfies* (i).

Proof: Since CP executes Step L.1 or Step R.1 depending on the condition checked at the beginning of an iteration, and the analysis in both cases is similar, we assume without loss of generality that Step L.1 is executed in this particular iteration of CP. Moreover, if $t_1 = r(\delta_{\mathrm{L}}(C_{s_0}))$ in Step L.1, then we are done, because $s_0 = s_1$ in such case.

By Lemma 4, each iteration of PLB moves the left border of the corresponding expansion, which implies that $r(\delta_{\mathrm{L}}(C_{s_1})) < r(\delta_{\mathrm{L}}(C_{s_1-1})) < \cdots < r(\delta_{\mathrm{L}}(C_{s_0}))$. Moreover, by Lemma 9, $r(\delta_{\mathrm{L}}(C_{s_0})) < l(\delta_{\mathrm{R}}(C_{s_0}))$ and Lemma 6 implies that $V_i \cap \delta(C_{s_1}) = V_i \cap \delta(C_{s_0})$ for each $i \geq r(\delta_{\mathrm{L}}(C_{s_0}))$, which gives

$$\delta_{\mathrm{R}}(C_{s_0}) = \delta_{\mathrm{R}}(C_{s_0+1}) = \cdots = \delta_{\mathrm{R}}(C_{s_1}). \quad (8)$$

Let first $i \in \{r(\delta_{\mathrm{L}}(C_{s_1})), \ldots, r(\delta_{\mathrm{L}}(C_{s_0}))\}$. By (ii') for C_{s_0}, and by Lemma 6

$$\omega(V_i \cap C_{s_1}) = \omega(V_i \cap \widetilde{C}_{s_0}) + \omega(V_i \cap V(\mathcal{B}_1)) \geq \sum_{p<i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{s_0})) + \omega(V_i \cap V(\mathcal{B}_1)). \quad (9)$$

By (iii) for C_{s_0}, the cut $r(\delta_{\mathrm{L}}(C_{s_0}))$ is a bottleneck of the branch $\mathcal{B}_{\mathrm{L}}(C_{s_0}, i)$. Thus, the weight of the cut $r(\delta_{\mathrm{L}}(C_{s_0}))$, that is $\omega(\delta_{\mathrm{L}}(C_{s_0}))$, is not greater than the weight of the cut i of the branch $\mathcal{B}_{\mathrm{L}}(C_{s_0}, i)$, which equals $\omega(\mathrm{Ext}(\mathcal{B}_{\mathrm{L}}(C_{s_0}, i)))$. Since $V_i \cap V(\mathcal{B}_1) = V_i \cap V(\mathcal{B}_{\mathrm{L}}(C_{s_0}, i))$, by Observation 2 and by (9) we obtain $\omega(V_i \cap C_{s_1}) \geq \omega(\delta_{\mathrm{L}}(C_{s_0}))$. By the fact that Step L.1 of CP executes, $\omega(\delta_{\mathrm{L}}(C_{s_0})) \geq \omega(\delta_{\mathrm{R}}(C_{s_0}))$. Since, by (8), $\omega(\delta_{\mathrm{R}}(C_{s_0})) = \omega(\delta_{\mathrm{R}}(C_{s_1}))$, we obtain that $\omega(V_i \cap C_{s_1}) \geq \omega(\delta_{\mathrm{R}}(C_{s_1}))$.

Let now $i \in \{r(\delta_{\mathrm{L}}(C_{s_0})), \ldots, l(\delta_{\mathrm{R}}(C_{s_1}))\}$. As argued above, $\omega(\delta_{\mathrm{R}}(C_{s_0})) \leq \omega(\delta_{\mathrm{L}}(C_{s_0}))$. Thus, by (8), by Lemma 6 and by (i) for C_{s_0}, $\omega(V_i \cap C_{s_1}) = \omega(V_i \cap C_{s_0}) \geq \omega(\delta_{\mathrm{R}}(C_{s_0})) = \omega(\delta_{\mathrm{R}}(C_{s_1}))$. □

Lemma 12 *If an expansion C_{s_0} from the beginning of an iteration of* CP *is nested, then each expansion computed by* CP *in this iteration satisfies* (ii).

Proof: As before, assume without loss of generality that CP executes Steps L.1-L.2 in the iteration we consider, and the proof for the other case is similar.

Due to Lemma 10, it is enough to prove that an expansion C_{s_2} obtained in Step L.2 of CP satisfies (ii). Note that $C_{s_1} = C_{s_2-1}$ is the expansion obtained at the end of Step L.1 of CP. If Step LE is executed at least once by PLB invoked in Step L.1 of CP, then, by Lemmas 10 and 11, C_{s_1} satisfies (i) and (ii), otherwise C_{s_1} is nested by assumption.

By the fact that $r(\delta_{\mathrm{L}}(C_{s_0}))$ is a bottleneck of $\mathcal{B}_1$ from Step L.1, we obtain that $\omega(\delta_{\mathrm{L}}(C_{s_1})) \geq \omega(\delta_{\mathrm{L}}(C_{s_0}))$. By assumption, $\omega(\delta_{\mathrm{L}}(C_{s_0})) \geq \omega(\delta_{\mathrm{R}}(C_{s_0}))$. By Lemma 6, $\delta_{\mathrm{R}}(C_{s_0}) = \delta_{\mathrm{R}}(C_{s_0+1}) = \cdots = \delta_{\mathrm{R}}(C_{s_1})$. Thus,

$$\omega(\delta_{\mathrm{L}}(C_{s_1})) \geq \omega(\delta_{\mathrm{R}}(C_{s_1})). \quad (10)$$

First we prove (ii) for the left border of C_{s_2}. (See Figure 8 for an example of C_{s_1} and C_{s_2}.) By construction, $r(\delta_{\mathrm{L}}(C_{s_2})) \leq t_1$. Since $C_{s_2} \subseteq C_{s_1} \cup V_{t_1+1}$, it holds $V_{t_1} \cap \delta_{\mathrm{L}}(C_{s_2}) \subseteq V_{t_1} \cap \delta_{\mathrm{L}}(C_{s_1})$ and $V_i \cap$

$\delta_{\mathrm{L}}(C_{s_2}) = V_i \cap \delta_{\mathrm{L}}(C_{s_1})$ for $i < t_1$. Moreover, $V_i \cap C_{s_2} = V_i \cap C_{s_1}$ for each $i \leq t_1$. Thus, for $i \leq t_1$, by (ii) for C_{s_1},

$$\omega(V_i \cap C_{s_2}) = \omega(V_i \cap C_{s_1}) \geq \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{s_1})) \geq \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{s_2})). \tag{11}$$

Now we analyze the right border of C_{s_2}. It holds $l(\delta_{\mathrm{R}}(C_{s_2})) \geq t_1 + 1$, because $\delta_{\mathrm{R}}(C_{s_2}) \subseteq \delta_{\mathrm{R}}(C_{s_1}) \cup V_{t_1+1}$ and $l(\delta_{\mathrm{R}}(C_{s_1})) \geq t_1 + 1$. Also,

$$V_i \cap C_{s_2} = V_i \cap C_{s_1} \text{ for each } i > t_1 + 1. \tag{12}$$

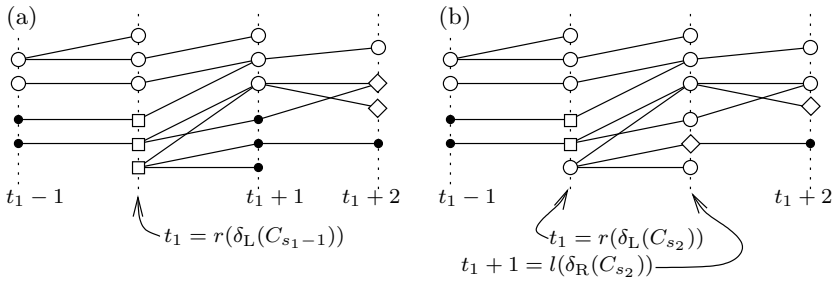

Figure 8: Proof of Lemma 12: (a) C_{s_1}; (b) C_{s_2}

Note that $\delta_{\mathrm{R}}(C_{s_2}) \subseteq \delta_{\mathrm{R}}(C_{s_1}) \cup V_{t_1+1}$ implies that if $l(\delta_{\mathrm{R}}(C_{s_2})) > t_1 + 1$, then $\delta_{\mathrm{R}}(C_{s_2}) \subseteq \delta_{\mathrm{R}}(C_{s_1})$ and consequently, by (12), C_{s_2} satisfies (ii). Thus, we continue with the assumption that $l(\delta_{\mathrm{R}}(C_{s_2})) = t_1 + 1$. Let $i \geq l(\delta_{\mathrm{R}}(C_{s_2}))$ be selected arbitrarily. We consider the following cases.

1. $i = l(\delta_{\mathrm{R}}(C_{s_2})) = t_1 + 1$. By the definition and by Lemmas 8 and 9, $V_i \cap \delta_{\mathrm{R}}(C_{s_2})$ and $V_i \cap \widetilde{C}_{s_2}$ are disjoint and their union is $V_i \cap C_{s_2}$. By construction, $V_i \cap \widetilde{C}_{s_2} \supseteq V_i \cap \widetilde{C}_{s_1}$ (see Figure 8 for an example where the equality between the sets does not hold), which implies

   $$\omega(V_i \cap \widetilde{C}_{s_2}) \geq \omega(V_i \cap \widetilde{C}_{s_1}). \tag{13}$$

   By Lemma 9, $l(\delta_{\mathrm{R}}(C_{s_1})) > t_1$, and $r(\delta_{\mathrm{L}}(C_{s_1})) \leq t_1$. Thus, we obtain by (i) for C_{s_1} that $\omega(V_i \cap C_{s_1}) \geq \min\{\omega(\delta_{\mathrm{L}}(C_{s_1})), \omega(\delta_{\mathrm{R}}(C_{s_1}))\}$. By (10), $\omega(V_i \cap C_{s_1}) \geq \omega(\delta_{\mathrm{R}}(C_{s_1}))$. By (ii') for C_{s_1} and by (13)

   $$\begin{aligned}
   \omega(V_i \cap C_{s_2}) &= \omega(V_i \cap \delta_{\mathrm{R}}(C_{s_2})) + \omega(V_i \cap \widetilde{C}_{s_2}) \\
   &\geq \omega(V_i \cap \delta_{\mathrm{R}}(C_{s_2})) + \omega(V_i \cap \widetilde{C}_{s_1}) \\
   &= \omega(V_i \cap (\delta_{\mathrm{R}}(C_{s_2}) \setminus \delta_{\mathrm{R}}(C_{s_1}))) + \omega(V_i \cap C_{s_1}) \\
   &\geq \omega(V_i \cap (\delta_{\mathrm{R}}(C_{s_2}) \setminus \delta_{\mathrm{R}}(C_{s_1}))) + \omega(\delta_{\mathrm{R}}(C_{s_1})) \\
   &= \omega(\delta_{\mathrm{R}}(C_{s_2})),
   \end{aligned}$$

   where the last equation follows from (12).

2. $l(\delta_{\mathrm{R}}(C_{s_2})) < i \leq l(\delta_{\mathrm{R}}(C_{s_1}))$. Since $i > t_1 + 1 > r(\delta_{\mathrm{L}}(C_{s_1}))$, again by (10),(12) and by (i),

   $$\omega(V_i \cap C_{s_2}) = \omega(V_i \cap C_{s_1}) \geq \omega(\delta_{\mathrm{R}}(C_{s_1})) \geq \sum_{p \geq i} V_p \cap \delta_{\mathrm{R}}(C_{s_2}).$$

3. $i > l(\delta_{\mathrm{R}}(C_{s_1}))$. Since C_{s_1} satisfies (ii), Equation (12) implies (ii) for C_{s_2}.

Cases 1-3 and (11) imply that (ii) holds for C_{s_2}. □

Lemma 13 *Let C_{s_0} and C_{s_3} be the expansions from the beginning of two consecutive iterations of* CP*. If C_{s_0} is nested, then C_{s_3} is nested.*

Proof: As in the previous proofs, we continue without loss of generality with the assumption that $\omega(\delta_{\mathrm{L}}(C_{s_0})) > \omega(\delta_{\mathrm{R}}(C_{s_0}))$ at the beginning of the iteration of CP we consider. Let C_{s_1} and C_{s_2} be the expansions obtained at the end of Steps L.1 and L.2 of CP, respectively.

First we prove that the expansion C_{s_3} satisfies (i). It holds $r(\delta_{\mathrm{L}}(C_{s_3})) \leq t'_3 \leq t_1 < t'_2 \leq l(\delta_{\mathrm{R}}(C_{s_3}))$ and $t_1 \leq r(\delta_{\mathrm{L}}(C_{s_0}))$ (see also Figure 4). This follows from the formulation of PLB and from the definition of a branch, because for each i, $t'_3 < i < t'_2$, $V_i \cap \delta(C_{s_3}) = \emptyset$, which is a consequence of the fact that $\mathcal{B}'_2$ and $\mathcal{B}'_3$ are proper.

Let $i \in \{r(\delta_{\mathrm{L}}(C_{s_3})), \ldots, l(\delta_{\mathrm{R}}(C_{s_3}))\}$. We consider several cases shown in Figure 9, where C_{s_0} with its left and right borders is given ($C_{s_1} = C_{s_0} \cup V(\mathcal{B}_1)$). Note that the example is constructed so that Case 1

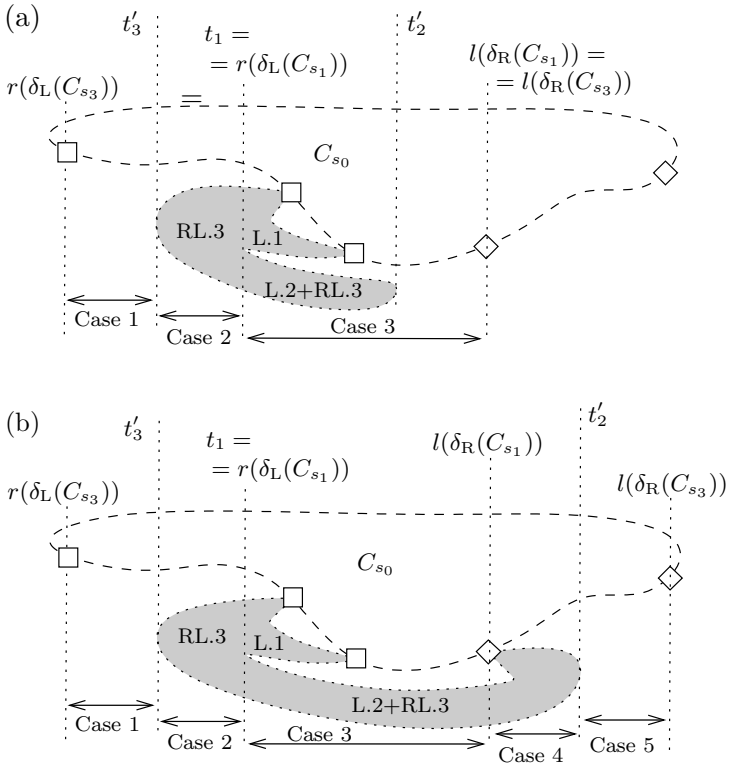

Figure 9: C_{s_0} together with its borders; (a) $t'_2 \leq l(\delta_{\mathrm{R}}(C_{s_1}))$; (b) $t'_2 > l(\delta_{\mathrm{R}}(C_{s_1}))$

and Case 5 (if applies) are 'non-empty', which occurs if the branches $\mathcal{B}'_p$ have no external vertices in $V_{t'_p}$, $p = 2, 3$. In each case we prove that $\omega(V_i \cap C_{s_3}) \geq \omega(\delta_{\mathrm{L}}(C_{s_3}))$ or $\omega(V_i \cap C_{s_3}) \geq \omega(\delta_{\mathrm{R}}(C_{s_3}))$.

Case 1: $r(\delta_{\mathrm{L}}(C_{s_3})) \leq i < t'_3$. By construction, $V_p \cap C_{s_3} = V_p \cap C_{s_0}$ for each $p < t'_3$, $V_p \cap \delta_{\mathrm{L}}(C_{s_3}) = V_p \cap \delta_{\mathrm{L}}(C_{s_0})$ for each $p < t'_3 - 1$ and $V_{t'_3-1} \cap \delta_{\mathrm{L}}(C_{s_3}) \subseteq V_{t'_3-1} \cap \delta_{\mathrm{L}}(C_{s_0})$. By (ii) for C_{s_0} (note that $i < t_1 \leq r(\delta_{\mathrm{L}}(C_{s_0}))$), $\omega(V_i \cap C_{s_0}) \geq \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{s_0}))$. Thus,

$$\omega(V_i \cap C_{s_3}) = \omega(V_i \cap C_{s_0}) \geq \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{s_3})). \tag{14}$$

The fact that $r(\delta_{\mathrm{L}}(C_{s_3})) \leq i$ implies

$$\bigcup_{p \leq i} V_p \cap \delta_{\mathrm{L}}(C_{s_3}) = \delta_{\mathrm{L}}(C_{s_3}),$$

which by (14) gives $\omega(V_i \cap C_{s_3}) \geq \omega(\delta_{\mathrm{L}}(C_{s_3}))$.

Case 2: $t'_3 \leq i \leq t_1$. By Lemma 6 and by the formulation of Step L.1 of CP, $V_i \cap C_{s_3} = (V_i \cap \widetilde{C}_{s_0}) \cup (V_i \cap V(\mathcal{B}'_3))$. Since $\widetilde{C}_{s_0}$ and $V(\mathcal{B}'_3)$ are (by the definition) disjoint, we obtain that $\omega(V_i \cap C_{s_3}) = \omega(V_i \cap \widetilde{C}_{s_0}) + \omega(V_i \cap V(\mathcal{B}'_3))$. By assumption, C_{s_0} satisfies (ii), and $i \leq t_1 \leq r(\delta_{\mathrm{L}}(C_{s_0}))$. Thus, by (ii') and by Observations 1 and 2,

$$\omega(V_i \cap C_{s_3}) \geq \sum_{p<i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{s_0})) + \omega(V_i \cap V(\mathcal{B}'_3)) \geq \mathrm{Ext}(\mathcal{B}_{\mathrm{L}}(C_{s_2}, i)) \geq \mathrm{Ext}(\mathcal{B}_{\mathrm{L}}(C_{s_2}, t'_3)) = \omega(\delta_{\mathrm{L}}(C_{s_3})),$$

because t'_3 is a bottleneck of $\mathcal{B}_3$.

Case 3: $t_1 < i \leq l(\delta_{\mathrm{R}}(C_{s_1}))$. By Lemma 6, by the choice of i and by the fact that $\mathcal{B}_2$, and therefore $\mathcal{B}'_2$, is proper, the set $V_i \cap C_{s_3}$ is an union of two disjoint sets $V_i \cap C_{s_1}$ and $V_i \cap V(\mathcal{B}'_2)$, which gives

$$\omega(V_i \cap C_{s_3}) = \omega(V_i \cap C_{s_1}) + \omega(V_i \cap V(\mathcal{B}'_2)). \tag{15}$$

By Lemma 11, C_{s_1} satisfies (i), which in particular gives $\omega(V_i \cap C_{s_1}) \geq \min\{\omega(\delta_{\mathrm{L}}(C_{s_1})), \omega(\delta_{\mathrm{R}}(C_{s_1}))\}$.

If $\omega(\delta_{\mathrm{L}}(C_{s_1})) \geq \omega(\delta_{\mathrm{R}}(C_{s_1}))$, then

$$\omega(V_i \cap C_{s_1}) + \omega(V_i \cap V(\mathcal{B}'_2)) \geq \omega(\delta_{\mathrm{R}}(C_{s_1})) + \omega(V_i \cap V(\mathcal{B}'_2)) \geq \omega(\delta_{\mathrm{R}}(C_{s_3})),$$

because t'_2 is a bottleneck of $\mathcal{B}_2$. By (15), $\omega(V_i \cap C_{s_3}) \geq \omega(\delta_{\mathrm{R}}(C_{s_3}))$.

If $\omega(\delta_{\mathrm{L}}(C_{s_1})) < \omega(\delta_{\mathrm{R}}(C_{s_1}))$, then by (15)

$$\omega(V_i \cap C_{s_3})) \geq \omega(\delta_{\mathrm{L}}(C_{s_1})) + \omega(V_i \cap V(\mathcal{B}'_2)) \geq \omega(\delta_{\mathrm{L}}(C_{s_3})) + \omega(V_i \cap V(\mathcal{B}'_2)) \geq \omega(\delta_{\mathrm{L}}(C_{s_3})),$$

where $\omega(\delta_{\mathrm{L}}(C_{s_1})) \geq \omega(\delta_{\mathrm{L}}(C_{s_3}))$ follows from the fact that t'_3 is a bottleneck of $\mathcal{B}_3$.

Case 4: $l(\delta_{\mathrm{R}}(C_{s_1})) < i \leq t'_2$. Note that if $l(\delta_{\mathrm{R}}(C_{s_1})) > t'_2$, then there is nothing to prove (see Figure 9(a)). Otherwise the situation is symmetric to Case 2 and (due to $\delta_{\mathrm{R}}(C_{s_1}) \subseteq \delta_{\mathrm{R}}(C_{s_0})$) leads to inequality $\omega(V_i \cap C_{s_3}) \geq \omega(\delta_{\mathrm{R}}(C_{s_3}))$.

Case 5: $t'_2 < i \leq l(\delta_{\mathrm{R}}(C_{s_3}))$. The proof is analogous to Case 1 and gives $\omega(V_i \cap C_{s_3}) \geq \omega(\delta_{\mathrm{R}}(C_{s_3}))$.

Since in each case we obtain that $\omega(V_i \cap C_{s_3}) \geq \omega(\delta_{\mathrm{L}}(C_{s_3}))$ or $\omega(V_i \cap C_{s_3}) \geq \omega(\delta_{\mathrm{R}}(C_{s_3}))$, where $i \in \{r(\delta_{\mathrm{L}}(C_{s_3})), \ldots, l(\delta_{\mathrm{R}}(C_{s_3}))\}$, we have proved (i) for C_{s_3}.

Lemma 12 implies that C_{s_3} satisfies (ii).

Note that (iii) for C_{s_3} follows directly from Step RL.3 of CP. Indeed, if t'_3 is not a bottleneck of any left branch $\mathcal{B}''_3 = \mathcal{B}_{\mathrm{L}}(C_{s_3}, i)$, i.e. $t''_3 < t'_3$ is its bottleneck, then t''_3 is also a bottleneck of $\mathcal{B}_3$ from Step RL.3 of CP (the union of $\mathcal{B}'_3$ and $\mathcal{B}''_3$ is a subgraph of $\mathcal{B}_3$, because $\mathcal{B}_3$ is maximal), which contradicts the choice of the branch $\mathcal{B}'_3$. Similar argument holds for the right border of C_{s_3}. □

Lemma 14 *It holds $\omega(B_j) \leq 2 \cdot \mathrm{width}(\mathcal{G})$ for each $j = 1, \ldots, m$.*

Proof: An expansion obtained at the end of Step I.2 of CP is nested. Indeed, its left border is empty which implies (i) and gives (ii) and (iii) for the left border. Condition (iii) for the right border follows from a'_0 being the bottleneck of the branch used in Step I.2, and (ii) trivially holds, because the right border is contained in a single set V_i. Therefore, using an induction (on the number of iterations of CP) we obtain by Lemma 13

that any expansion from the beginning of an iteration of CP is nested. Thus, Lemma 12, implies that C_j satisfies (ii) for each $j = 1, \ldots, m$.

Let $j \in \{2, \ldots, m\}$. First note that $\max\{\omega(\delta_{\mathrm{R}}(C_j)), \omega(\delta_{\mathrm{L}}(C_j))\} \leq \mathrm{width}(\mathcal{G})$. Indeed, by (ii), where $i = l(\delta_{\mathrm{R}}(C))$,

$$\omega(V_i \cap C_j) \geq \sum_{p \geq i} \omega(V_p \cap \delta_{\mathrm{R}}(C_j)) = \omega(\delta_{\mathrm{R}}(C_j)). \tag{16}$$

Since $\omega(V_i \cap C_j) \leq \omega(V_i) \leq \mathrm{width}(\mathcal{G})$, we obtain $\omega(\delta_{\mathrm{R}}(C_j)) \leq \mathrm{width}(\mathcal{G})$. Analogously one can prove that $\omega(\delta_{\mathrm{L}}(C_j)) \leq \mathrm{width}(\mathcal{G})$.

Now we give the upper bound on $\omega(B_j)$. The claim clearly follows for $j = 1$ so assume in the following that $j > 1$. By construction, $B_j = \delta(C_j) \cup A_j$. Thus, due to Lemma 8 we obtain that if

$$\omega(A_j \cup \delta_{\mathrm{L}}(C_j)) \leq \mathrm{width}(\mathcal{G}) \text{ or } \omega(A_j \cup \delta_{\mathrm{R}}(C_j)) \leq \mathrm{width}(\mathcal{G}), \tag{17}$$

then, by (16), the lemma follows, so we now prove that one of those inequalities holds.

By construction, $A_j \subseteq V_i$ for some $i \in \{1, \ldots, d\}$, and

$$A_j \cap (V_i \cap C_{j-1}) = \emptyset, \tag{18}$$

because $A_j \cap C_{j-1} = \emptyset$. We consider two cases:

Case 1: C_j has been computed in Step L.2 or in Step R.2 of CP. Assume without loss of generality that the former occurs. By construction, $i \in \{r(\delta_{\mathrm{L}}(C_{j-1})), \ldots, l(\delta_{\mathrm{R}}(C_{j-1}))\}$ and, by Lemma 11, C_{j-1} satisfies (i). Hence, by (18),

$$\min\{\omega(A_j \cup \delta_{\mathrm{L}}(C_{j-1})), \omega(A_j \cup \delta_{\mathrm{R}}(C_{j-1}))\} \leq \omega(A_j \cup (V_i \cap C_{j-1})) \leq \omega(V_i) \leq \mathrm{width}(\mathcal{G}). \tag{19}$$

By the formulation of Step LE, $\delta_{\mathrm{L}}(C_j) \subseteq A_j \cup \delta_{\mathrm{L}}(C_{j-1})$, which gives $A_j \cup \delta_{\mathrm{L}}(C_j) \subseteq A_j \cup \delta_{\mathrm{L}}(C_{j-1})$. Similarly, $A_j \cup \delta_{\mathrm{R}}(C_j) \subseteq A_j \cup \delta_{\mathrm{R}}(C_{j-1})$. Consequently, $\omega(A_j \cup \delta_{\mathrm{L}}(C_j)) \leq \omega(A_j \cup \delta_{\mathrm{L}}(C_{j-1}))$ and $\omega(A_j \cup \delta_{\mathrm{R}}(C_j)) \leq \omega(A_j \cup \delta_{\mathrm{R}}(C_{j-1}))$. Equation (19) implies (17), which gives the desired bound on $\omega(B_j)$.

Case 2: C_j has been computed by PLB or PRB. Since both cases are analogous, assume again that the former occurs. By Lemma 4, C_j moves the left border of C_{j-1}, which gives that $i \leq r(\delta_{\mathrm{L}}(C_{j-1}))$. Moreover, $\delta_{\mathrm{L}}(C_j) \subseteq A_j \cup \bigcup_{p \leq i} V_p \cap \delta_{\mathrm{L}}(C_{j-1})$. Thus, $A_j \cup \delta_{\mathrm{L}}(C_j) \subseteq A_j \cup \bigcup_{p \leq i} V_p \cap \delta_{\mathrm{L}}(C_{j-1})$, and therefore

$$\omega(A_j \cup \delta_{\mathrm{L}}(C_j)) \leq \omega(A_j) + \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{j-1})).$$

Then, by (18) and by (ii) for C_{j-1},

$$\omega(A_j) + \sum_{p \leq i} \omega(V_p \cap \delta_{\mathrm{L}}(C_{j-1})) \leq \omega(A_j) + \omega(V_i \cap C_{j-1}) \leq \omega(V_i) \leq \mathrm{width}(\mathcal{G}),$$

and (17) follows.

□

Lemma 15 *If $\mathcal{C} = (Z_1, \ldots, Z_m)$ is a path decomposition calculated by* CP *for the given G and $\mathcal{P}$, then* $\mathrm{width}(\mathcal{C}) \leq 2 \cdot \mathrm{width}(\mathcal{P}) + 1$.

Proof: By Lemma 14, $\omega(B_j) \leq 2 \cdot \mathrm{width}(\mathcal{G})$. By the definition, $\mathrm{width}(\mathcal{C}) = \max\{\omega(B_j) \colon j = 1, \ldots, m\} - 1$. Thus, by the definition, $\mathrm{width}(\mathcal{C}) \leq 2 \cdot \mathrm{width}(\mathcal{G}) - 1 = 2 \cdot \mathrm{width}(\mathcal{P}) + 1$. □

Lemma 16 *Let G be a simple connected graph and let $\mathcal{P} = (X_1, \ldots, X_d)$ be its path decomposition of width k. The running time of* CP *executed for G and $\mathcal{P}$ is $O(dk^2)$.*

Proof: Since each edge of G is contained in one of the bags of $\mathcal{P}$, $|E(G)| \leq dk^2$. The number of vertices and edges in $\mathcal{G}$ is $O(kd)$ and $O(dk^2)$, respectively. Thus, the complexity of constructing $\mathcal{G}$ is $O(dk^2)$.

Here we assume that each branch $\mathcal{B}$ used by CP is encoded as a linked list such that each element of this list is a non-empty set $V_i \cap V(\mathcal{B})$, $i \in \{1, \ldots, d\}$. If a branch is given, then the weights of all its cuts can be calculated in time linear in the number of edges and vertices of the branch. The computation of a branch is done by the execution of the procedure PLB or PRB (due to symmetry assume that the former occurs), and if C_j and $C_{j'}$ are the expansions from the beginning and from the end, respectively, of a particular execution of PLB, then the vertex set of the branch is $\delta_{\mathrm{L}}(C_j) \cup (C_{j'} \setminus C_j)$ for the corresponding left branch. The time of finding any branch $\mathcal{B}$ in an iteration of CP is therefore $O(|E(\mathcal{B})|)$. Also note that, while recording a subset $A_j \subseteq V_i$ of vertices of $\mathcal{B}$ during the execution of PLB, the weight of cut i of the corresponding branch can be efficiently obtained, because it is equal to $\omega(\delta_{\mathrm{L}}(C_j))$. Moreover, for each $j' > j$ if $C_{j'}$ has been computed in the same execution of PLB, then the weight of cut i of the branch corresponding to $C_{j'}$ remains $\omega(\delta_{\mathrm{L}}(C_j))$. Thus, the complexity of calculating the weight of all cuts of $\mathcal{B}$, and thus finding its bottleneck, is $O(|E(\mathcal{B})|)$.

Whenever two branches overlap, we do not have to repeat the computation, because due to Observation 1, one is contained in the other, and their vertex sets and cuts are identical for common induces i. Therefore, the time complexity of determining all branches and their bottlenecks is $O(dk^2)$. This includes the complexity of all executions of the procedures PLB and PRB, because, by Lemma 6, the procedure 'follows' the previously calculated branches by including their vertices into the expansions C_j. It holds that $m \leq kd$, because (by Lemma 6 and by the formulation of Steps LE and RE) $C_j \subseteq C_{j+1}$ and $C_j \neq C_{j+1}$ for each $j = 1, \ldots, m-1$. By Lemma 15, $\omega(B_j) = O(k)$ for each $j = 1, \ldots, m$. Thus, $\sum_{1 \leq j \leq m} |Z_j| = O(dk^2)$. Therefore, the overall complexity of CP is $O(dk^2)$. □

Theorem 1 *There exists an algorithm that for the given connected graph G and its path decomposition $\mathcal{P} = (X_1, \ldots, X_d)$ of width k returns a connected path decomposition $\mathcal{C} = (Z_1, \ldots, Z_m)$ such that* $\mathrm{width}(\mathcal{C}) \leq 2k + 1$ *and $m \leq kd$. The running time of the algorithm is $O(dk^2)$.*

Proof: The correctness of the algorithm CP is due to Lemma 7. The inequality $\mathrm{width}(\mathcal{C}) \leq 2 \cdot \mathrm{width}(\mathcal{P}) + 1$ follows from Lemma 15, while the complexity of CP is due to Lemma 16. As argued in the proof of Lemma 16, $m \leq kd$. □

Theorem 2 *For each connected graph G,* $\mathrm{cpw}(G) \leq 2 \cdot \mathrm{pw}(G) + 1$. □

7 Applications in graph searching

In this section we restate the main result of the previous section in terms of the graph searching numbers. In addition, we propose a small modification to the algorithm CP which can be used to convert a search strategy into a connected one that starts at an arbitrary homebase $h \in V(G)$. To that end it is sufficient to guarantee that h belongs to the first bag of the resulting path decomposition $\mathcal{C}$. The modification changes only the initialization stage of CP.

Consider the following procedure CPH (*Connected Pathwidth with Homebase*) obtained by replacing the Steps I.1-I.2 of CP with Steps I.1'-I.3':

Procedure CPH (*Connected Pathwidth with Homebase*)

Input: A simple graph G, a path decomposition $\mathcal{P}$ of G, and $h \in V(G)$.

Output: A connected path decomposition $\mathcal{C}$ of G with h in its first bag.

begin

I.1': Use G and $\mathcal{P}$ to calculate the derived graph $\mathcal{G}$. Let $v = v(H)$ be any vertex of $\mathcal{G}$ such that $h \in V(H)$. Let $C_1 = \{x, y\}$, where $v \in C_1$, x, y are adjacent in $\mathcal{G}$, and $x \in V_i$, $y \in V_{i+1}$ for some $i \in \{1, \ldots, d-1\}$. Let $\delta_{\rm L}(C_1) = \{x\} \cap \delta(C_1)$, $\delta_{\rm R}(C_1) = \{y\} \cap \delta(C_1)$ and let $m = 1$.

I.2': If $\delta_{\rm L}(C_m) \neq \emptyset$, then find the maximal left branch $\mathcal{B}_{\rm L}(C_m, a_0)$ with a bottleneck a'_0 ($a'_0 \geq a_0$) and call $\texttt{PLB}(a'_0)$.

I.3': If $\delta_{\rm R}(C_m) \neq \emptyset$, then find the maximal right branch $\mathcal{B}_{\rm R}(C_m, b_0)$ with a bottleneck b'_0 ($b'_0 \leq b_0$) and call $\texttt{PRB}(b'_0)$.

Execute the main loop of CP*, compute* $Z_1, \ldots, Z_m$ *as in* CP *and return* $\mathcal{C}$.

End procedure CPH.

Lemma 17 *Given a simple graph G, a path decomposition $\mathcal{P}$ of G and $h \in V(G)$ as an input, the algorithm* CPH *returns a connected path decomposition $\mathcal{C} = (Z_1, \ldots, Z_m)$ of G such that* $\mathrm{width}(\mathcal{C}) \leq 2 \cdot \mathrm{width}(\mathcal{P}) + 1$ *and $h \in Z_1$.*

Proof: Note that $h \in Z_1$ follows from Step I.1' of CPH. If C_j is an expansion obtained in one of Steps I.1'-I.3', then $B_j \subseteq V_i \cup V_{i'}$ for some $i, i' \in \{1, \ldots, d\}$. Thus, $|Z_j| \leq 2 \cdot \mathrm{width}(\mathcal{G})$. The expansion obtained at the end of Step I.3' is nested, which follows from the fact that a'_0 and b'_0 are the bottlenecks of the branches computed in Steps I.2' and I.3'. Thus, the proof is analogous to the proof of Lemma 15. □

Suppose that we are given a graph G, a search strategy that uses k searchers, and a vertex $h \in V(G)$ that is required to be the homebase of the connected search strategy to be computed. Recall that $\texttt{ns}(G) = \mathrm{pw}(G)+1$, and whatsmore, a node search strategy for G that uses p searchers can be converted into a path decomposition of G of width $p-1$ and vice versa (see [5, 15, 16, 21]). Thus, the initial (non-connected) search strategy can be converted into a path decomposition $\mathcal{P}$ of width k [17]. By Lemma 17, the procedure CPH returns a connected path decomposition $\mathcal{C}$ of width $2k+1$, where k is the width of the input path decomposition $\mathcal{P}$. Moreover, the homebase h is guaranteed to belong to the first bag Z_1 of $\mathcal{C}$. Then, we convert $\mathcal{C}$ into a monotone and connected search strategy that uses at most $2k+3$ searchers [17]. The monotonicity follows directly from the definition of path decomposition. This leads to the following.

Theorem 3 *There exists an algorithm that for a given connected graph G, $h \in V(G)$ and a search strategy that uses k searchers returns a monotone connected search strategy with homebase h that uses at most $2k+3$ searchers. The running time of the algorithm is $O(dk^2)$.* □

Theorem 4 *For each connected graph G,* $\texttt{cs}(G) \leq \texttt{mcs}(G) \leq 2 \cdot \texttt{s}(G) + 3$. □

8 Conclusions

The advances in graph theory presented in this paper are three-fold:

- A bound for connected pathwidth is given, $\mathrm{cpw}(G) \leq 2 \cdot \mathrm{pw}(G)+1$, where G is any graph, which bounds the connected search number of a graph, $\texttt{cs}(G) \leq 2\texttt{s}(G) + 3$. Moreover, a vertex v that belongs to the first bag in the resulting connected path decomposition can be selected arbitrarily, which implies a stronger fact, namely a connected $(2\texttt{s}(G)+3)$-search strategy can be constructed with any vertex of G playing the role of the homebase. Since one can obtain a path decomposition of width k for a given search strategy that uses k searchers, our algorithm provides an efficient algorithm for converting a search strategy into a connected one, and in addition, the homebase in the latter one can be chosen arbitrarily.
- An efficient method is given for calculating a connected path decomposition of width at most $2k+1$, provided that a graph G and its path decomposition of width k are given as an input.

- It is a strong assumption that the algorithm requires a path decomposition to be given, because calculating $\mathrm{pw}(G)$ is a hard problem even for graphs G that belong to some special classes of graphs. However, this algorithm can be used to approximate the connected pathwidth, because any $O(q)$-approximation algorithm for pathwidth can be used together with the algorithm from this paper to design a $O(q)$-approximation algorithm for connected pathwidth.

Acknowledgment. Thanks are due to the anonymous referees for constructive and insightful comments that helped to improve this work.

References

[1] B. Alspach. Searching and sweeping graphs: a brief survey. *Le Matematiche (Catania)*, 59:5–37, 2004.

[2] L. Barrière, P. Flocchini, F.V. Fomin, P. Fraigniaud, N. Nisse, N. Santoro, and D.M. Thilikos. Connected graph searching. Research Report RR-7363, INRIA, 2010.

[3] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *SPAA '02: Proc. of the fourteenth annual ACM symposium on parallel algorithms and architectures*, pages 200–209, New York, NY, USA, 2002. ACM.

[4] L. Barrière, P. Fraigniaud, N. Santoro, and D.M. Thilikos. Searching is not jumping. In *WG '03: Proc. of the 29th Inter. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 34–45, 2003.

[5] D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 5:33–49, 1991.

[6] D. Bienstock and P. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991.

[7] H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–22, 1993.

[8] D. Dereniowski. Connected searching of weighted trees. *Theor. Comp. Sci.*, 412:5700–5713, 2011.

[9] F.V. Fomin, P. Fraigniaud, and D.M. Thilikos. The price of connectedness in expansions. Technical report, Technical Report, UPC Barcelona, 2004.

[10] F.V. Fomin and D.M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[11] P. Fraigniaud and N. Nisse. Connected treewidth and connected graph searching. In *Proc. of the 7th Latin American Symposium on Theoretical Informatics (LATIN'06), LNCS*, volume 3887, pages 479–490, Valdivia, Chile, 2006.

[12] P. Fraigniaud and N. Nisse. Monotony properties of connected visible graph searching. *Inf. Comput.*, 206(12):1383–1393, 2008.

[13] J. Gustedt. On the pathwidth of chordal graphs. *Discrete Appl. Math.*, 45(3):233–248, 1993.

[14] T. Kashiwabara and T. Fujisawa. Np-completeness of the problem of finding a minimum-clique-number interval graph containing a given graph as a subgraph. In *Proc. IEEE Inter. Symp. Circuits and Systems*, pages 657–660, 1979.

[15] N.G. Kinnersley. The vertex separation number of a graph equals its path-width. *Inf. Process. Lett.*, 42(6):345–350, 1992.

[16] L.M. Kirousis and C.H. Papadimitriou. Interval graphs and searching. *Discrete App. Math.*, 55:181–184, 1985.

[17] L.M. Kirousis and C.H. Papadimitriou. Searching and pebbling. *Theor. Comput. Sci.*, 47(2):205–218, 1986.

[18] A.S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993.

[19] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, and C.H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35(1):18–44, 1988.

[20] R. Mihai and I. Todinca. Pathwidth is NP-hard for weighted trees. In *FAW '09: Proc. of the 3rd Inter. Workshop on Frontiers in Algorithmics*, pages 181–195, Berlin, Heidelberg, 2009. Springer-Verlag.

[21] R. Möhring. Graph problems related to gate matrix layout and PLA folding. In *E. Mayr, H. Noltemeier, and M. Syslo eds, Computational Graph Theory, Computing Supplementum*, volume 7, pages 17–51, 1990.

[22] N. Nisse. Connected graph searching in chordal graphs. *Discrete Applied Math.*, 157(12):2603–2610, 2008.

[23] S.L. Peng, M.T. Ko, C.W. Ho, T.S. Hsu, and C.Y. Tang. Graph searching on chordal graphs. In *ISAAC '96: Proc. of the 7th Inter. Symposium on Algorithms and Computation*, pages 156–165, London, UK, 1996. Springer-Verlag.

[24] N. Robertson and P.D. Seymour. Graph minors. I. excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61, 1983.

[25] N. Robertson and P.D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[26] B. Yang, D. Dyer, and B. Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770–5780, 2009.